福　井　大　學

学位論文［博士（工学）］

A Dissertation Submitted to the University of Fukui for the Degree of Doctor of Engineering

A Research of Evolutionary Computation for Combinatorial Optimization Problems

（組合せ最適化問題に対する進化的手法の研究）

平成 23 年 9 月

張　　亜 龍

（指導教官：小倉久和 教授）

Abstract

In the past decades, evolutionary computation has been developed as a new one of the intelligent computation technologies, which has received more and more attentions from various fields. In the industry engineering fields, there are many combinatorial optimization problems even operations research problems to be optimized by evolutionary computation. In the research of evolutionary computation to solve the combinatorial optimization problems, there were many satisfactory achievements which have been widely applied in practice. However as a young intelligent processing technology, evolutionary computation has not yet been enough for perfection and maturity, many factors impacting the performance of computation should be studied and improved further.

In solving the combinatorial optimization problems with Genetic Algorithms (GA), the one of the factors affecting the performance of algorithm is large number of lethal chromosomes which were generated in the population. Lethal chromosomes would be abandoned in generally which were decoded as infeasible solutions. However the lethal chromosomes were generated by those parents that were excellent individuals, they contain some excellent schemas. In this study, we propose two different methods to handle the lethal chromosomes for GA solving the different problems.

For GA solving the Multi-Dimensional Knapsack Problems (MDKP), all the lethal chromosomes, 0-1 character strings, are collected and cumulative into a multi-value character string named as vaccine. Although vaccine is created from lethal chromosomes, its quality could be proved being excellent. With that, we set up an algorithm model, double islands model, for GA recycling and using the lethal chromosomes. The process of recycling and handling the lethal chromosomes is called as immune operation in this study. Therefore, the proposed method is named Immune Genetic Algorithm (IGA) described as chapter 3.

Also, GA faces the problem of lethal chromosomes in solving the Multi-Knapsack Problems (MKP). The idea of GA handling the lethal chromosomes in solving the MDKP could also be applied flexibly in solving the MKP even if the chromosomes are multi-value encoding, but we try to propose another method in this study. To hand a lethal chromosome, an excellent non-lethal chromosome is firstly chosen by roulette method form population. Referring the loading program which is provided by selected excellent chromosome, some genes of lethal chromosome are reset so that the overloading knapsacks are removed some objects. We also called this process as immune operation, so this method is also called IGA which is described in chapter 4, but it is practically different from last one.

The representation how to encode the characteristics of solution as individual of chromosome and decoding it into solution again is another factor impacting the performance in evolutionary computation solving the combinatorial optimization problems. As another application of GA, this study proposes a representation for GA solving the two-dimensional orthogonal cutting problems with guillotine cutting requirement. A kind of 2-rows coding is designed in which each row contains two character strings. Decoder function is actually a heuristic process which could achieve the local optimal searching. Combining the global searching of GA, proposed algorithm obtains a satisfactory effectiveness as described in chapter 5.

In order to improve further the evolutionary performance in solving the two-dimensional orthogonal cutting problems, an application of Fish Swarm Optimization (FSO) method is proposed in this study described as chapter 6. With analyzing the relationship between the individuals, this study redesigns the definition of center function for FSO which is used to get the center position. In order to improve the speed of iteration, a new rule is adopted to select the behavior operation for every time of individual iterating. By this way, algorithm avoids from frequently performing the center function, the time

complexity of the algorithm is greatly reduced especially to large scale problems. Applying this application of FSO on large number of instances and comparing with GA, the experiments results shows that these designs could improve the effectiveness, convergence rate and stability for FSO.

Since various optimization algorithms take the different speed of operation or evolutionary (iteration) process, it is hard to measure the evolutionary curves against to generation or iteration. In this study most of evolutionary curves are given against to CPU-time. By this way, it is demonstrated clearly for various evolutionary computations to compare with the evolutionary process each other.

Contents

Chapter 1

Introduction

Computer science includes many technical fields; one of them is artificial intelligence. Evolutionary computation belongs to a branch of artificial intelligence in computer science, which uses the iterative approach to guide the optimization function to evolve forward the way of optimization solution. This approach is closely related to mechanism of evolution, therefore it is called “evolutionary computation”. Actually evolutionary computation includes techniques not only evolutionary algorithm but also other artificial intelligence techniques, which is often ignored by many literatures. Even some literatures refer the evolutionary algorithm as evolutionary computation; really they are not the same thing. This chapter references the latest researches and developments of computer science and artificial intelligence techniques, summarizes the evolutionary computation that mainly includes the evolutionary algorithms, swarm intelligence, self-organization theory, artificial intelligence system and other techniques.

1.1 Current Status of Evolutionary Computations

The meaning and main contents of evolutionary computation is summarized and described as Fig.1.1. In this structure about evolutionary computation, the thesis studied the evolutionary algorithm with Genetic Algorithm (GA), in which GA combines the technique of artificial, immune system, to solve the problem of lethal chromosomes, and applied this study to multi-dimensional knapsack problem (MKP) and Multi-Dimensional Knapsack Problem (MDKP). Also, the thesis studies the GA to solve the layout stock problems. In the last, the thesis also introduces the swarm intelligence algorithms, and proposed an application of fish warm optimization algorithm to solve the stock layout problems and compared the results with that of GA.

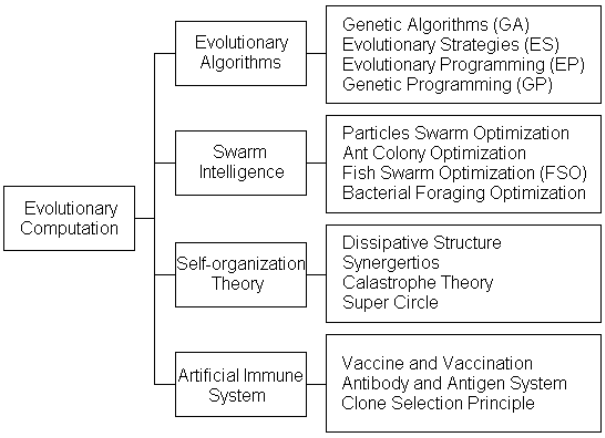

Fig. 1.1 Evolutionary computation

1.1.1 Evolutionary Algorithms

1960s, Prof. J. Holland of University of Michigan and his students proposed the GA [1] [2]. In the later the GA was gradually accepted by people and become to being popular in solving many optimization problems. I. Rechenberg and H. P. Schwefel of Technische Universität Berlin optimized the shape of objects with the ideal of biological variation at the beginning, but this technique gradually was developed as a set of Evolutionary Strategies (ES) [3]. Also there were appeared Evolutionary Programming (EP) and Genetic Programming (GP). This type of algorithms, GA, ES, EP and GP, is referred as Evolutionary Algorithms (EA).

EA is a set of techniques with the common features that they are all inspired by natural evolution of selection and random changes. These two elements can be translated to computers in two different ways. Computers can be used to simulate the evolution of species, but these elements can also used to build computer system that, following the principles of nature evolution established by Darwin (1859), evolve to optimize a function. Mathematical population genetics has used the first approach since the 1960s. The second approach has been used extensively by the computer science community during the last two decades. This is mainly in the field of optimization. Simulating this process on a computer could result in stochastic optimization techniques that can often outperform classical method of optimization when applied to difficult real-world problems.

Comparing with other algorithms, EA has distinctive features that are swarm and evolution, which greatly inspires the people's imagination. EC simulates the process of biological evolution to solve the problems skillfully, wherefore it has received much attention and was studied and developed by many researchers.

EA has some mainly features. (i) Randomness: stochastic optimization algorithm can solve the global nonlinear system optimization problems. (ii) Adaptive: adaptive method can solve the problem of machine learning. (iii) Parallelism: parallel algorithm has high computational efficiency. These three features make the researches and application of EA rapidly becoming the focus of attention at international academic and engineering.

Form 1990s, EA draws more and more attention as its distinctive features and broad prospects for development. Also as there are some defects and development can be improved in EA, constantly a variety of improved algorithms are proposed for EA. EA has not stopped the pace of the development. However the defects of EA cannot be ignored. At early stage of EA developing, the precocious problem has been studied by many researchers. Another important problem is about computational efficiency. For these defects, many improved algorithm have been proposed by researchers.

1.1.2 Swarm Intelligence Optimization

Swarm intelligence (SI), which is an artificial intelligence (AI) discipline, is concerned with the design of intelligent multi-agent systems by taking inspiration from the collective behavior of social insects such as ants, termites, bees, and wasps, as well as from other animal societies such as flocks of birds or schools of fish. Colonies of social insects have fascinated researchers for many years, and the mechanisms that govern their behavior remained unknown for a long time. Even though the single members of these colonies are non-sophisticated individuals, they are able to achieve complex tasks in cooperation. Coordinated colony behavior emerges from relatively simple actions or interactions between the colonies' individual members. Many aspects of the collective activities of social insects are self-organized and work without a central control. For example, leafcutter ants cut pieces from leaves, bring them back to their nest, and grow fungi used as food for their larvae. Weaver ant workers build chains with their bodies in order to cross gaps between two leaves. The edges of the two leaves are then pulled together, and successively connected by silk that is emitted by a mature larva held by a worker. Another example concerns the recruitment of other colony members for prey retrieval.

Other examples include the capabilities of termites and wasps to build sophisticated nests, or the ability of bees and ants to orient themselves in their environment. The term swarm intelligence was first used by Beni in the context of cellular robotic systems where simple agents organize themselves through nearest-neighbor interaction [4]. Swarm intelligence methods have been very successful in the area of optimization, which is of great importance for industry and science. This thesis aims at giving an application of fish warm optimization algorithm for stocks layout problems.

Optimization problems are of high importance both for the industrial world as well as for the scientific world. Examples of practical optimization problems include train scheduling, timetabling, shape optimization, telecommunication network design, and problems from computational biology. The research community has simplified many of these problems in order to obtain scientific test cases such as the well-known Multi-Dimensional Knapsack Problems (MDKP) and large scale stocks layout problem.

1.2 Main Contents of the Thesis

With summarizing the evolutionary computation as evolutionary algorithms, swarm intelligence, self-organization theory, artificial intelligence system and other techniques, this thesis firstly introduces the evolutionary algorithm with genetic algorithm, and analyzes the problem of lethal chromosomes in GA. Secondly, the thesis introduces the idea of artificial immune system into GA to solve the problem of lethal chromosomes, and proposed Immune Genetic Algorithm (IGA) working on a double islands model. Thirdly, to different Multi-Dimensional Knapsack Problem (MDKP) and Multi-Knapsack Problem (MKP), thesis proposes the different IGA to improve the performance of GA. Fourth, thesis proposes a GA to solve the stock layout problems and give some examples for layout pattern comparing with other algorithms. Fifth, thesis propose a analysis and application of fish swarm optimization also to stock

layout problems, and compares the testing results with that of GA. In the last thesis give discussions and conclusions with some summaries and outlooks for evolutionary computation to combinatorial optimization problems.

1.3 Structure of the Thesis

In this study, main contents consist of 8 chapters. Chapter 1 introduces the current status of evolutionary computation, and summarizes the main contents with some fields of artificial intelligence. In these fields, chapter 2 firstly introduces the genetic algorithm and fish swarm optimization algorithm, and then introduces the multi-dimensional knapsack problem (MDKP), Multi-Knapsacks problem (MKP) and stock layout problem, finally summaries the combinatorial optimization problems with a list. Chapter 3 analyzes the impact of lethal chromosomes to GA, proposes an immune genetic algorithm (IGA) based on a double islands model to solve the problem of lethal chromosomes, and applies the proposed algorithm to MDKP. Chapter 4 proposes another different immune genetic algorithm and applies it to MKP. Chapter 5 analyzes the application requirements of stock layout problems in practice production, proposes a genetic algorithm to solve the large scale stock layout problems, and compares the layout patterns with other algorithms. Chapter 6 proposes an application of swarm intelligence, fish swarm optimization, also apply to stock layout problems, and compare with proposed GA. Chapter 7 discusses GA and the lethal chromosomes, fish swarm optimization and other problems need to be improved, and then the application of combinatorial optimization problems. In the last, chapter 8 summarizes the conclusion of the thesis and future work, discusses the prospects of evolutionary computation.

Chapter 2

Evolutionary Computations and Combinatorial Optimization Problems

This thesis focuses on the problems of evolutionary computation to solve the combinatorial optimization problems, in which the lethal chromosomes of genetic algorithm and behavioral operation of fish swarm optimization are involved. On the other hand, the thesis is about the combinatorial optimization problems, which is inevitably involved in this thesis. In order to facilitate description for later chapters about problems involved, this chapter above all introduces some basics about evolutionary computation and some combinatorial optimization problems which are involved in this thesis.

2.1 Evolutionary Computations

Evolutionary computation includes many artificial intelligence technologies in computer science. In this large field, the thesis focuses on some problem points which involve the genetic algorithm and fish swarm optimization, this section introduce the basics of them respectively.

2.1.1 Genetic Algorithms

(1) *Brief Overview*

Genetic Algorithm (GA) is adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GA is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

First pioneered by John Holland in the 60s [1] [2], Genetic Algorithms has been widely studied, experimented and applied in many fields in engineering worlds. Not only does GA provide an alternative method to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GA. However, because of its outstanding performance in optimization, GA has been wrongly regarded as a function optimizer. In fact, there are many ways to view genetic algorithms. Perhaps most users come to GA looking for a problem solver, but this is a restrictive view. This thesis focuses on GA solving the combinatorial optimization problems, and tries to research the problem of lethal chromosomes for GA on MDKP and MKP.

(2) *Definition of GA*

GA was introduced as a computational analogy of adaptive systems. They are modeled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators such as recombination (crossover) and mutation. A fitness function is used to evaluate individuals, and reproductive success varies with fitness.

GA works to solve the problems from a population which consists of some chromosomes which are also called genes in some literatures; this thesis think chromosomes is more definite because that the genes constitute the chromosomes. Anyway they are a type of encoding reflecting the feature information of solutions, and constitute the population as individuals. Every individual actually is the entity of chromosome with feature information of solution. In reality, chromosome as main entity of genetic factors, that is combination of multiple genes, its internal performance is the combination of multiple genes but that decides the external performance of individual shapes. For example, characteristic of people with black hair is decided by some combination of genes in chromosomes.

The beginning of GA working to solve the problems is to design the encoding system. According to the encoding system, the initial population is generated which consist of some individuals. Based on survival of the fittest, the population evolves generation by generation until meeting the termination condition of GA. In every generation, GA distributes the chance to reproduce according to the fitness for individual, and evolves to the next generation via the

genetic operations selection, crossover and mutation. The evolution of population lead the individuals is improved to more adapt the living environment. GA obtains the optimal individual in the last generation to decode and then get the optimal solution of problem. This process of GA working is described as Fig.2.1.

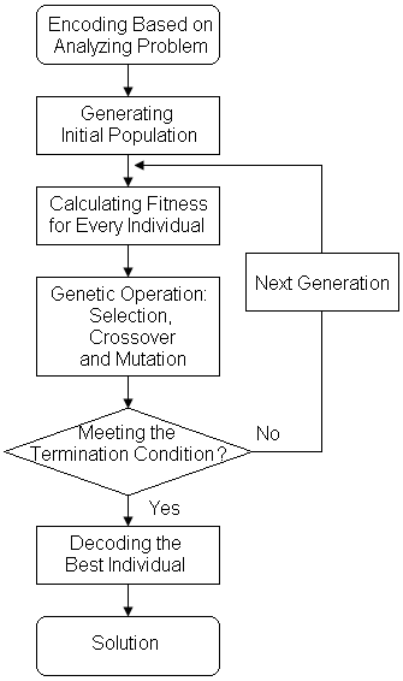

Fig. 2.1 Flowchart of the GA.

There are some cases could be the termination condition in GA.

(i) The running over the given generations.

(ii) The best individual obtained in population meets the given requirement.

(iii) The best individual (or mean fitness of population) has not improvement over several generations.

Schema Theorem and *Building Block Hypothesis* are the basic theory of the GA.

(i) *Schema Theorem*: Under the function of genetic operations selection, crossover and mutation, schemata with above-average fitness (especially short, low order schemata), increase their frequency in the population each generation at an exponential rate when rare.

(ii) *Building Block Hypothesis*: In the evolution of GA, schema with above-average fitness (especially short, low order schema) could combines together each other and become to the long, high order schemata with above-average fitness. Finally, they form the optimal solution of problem.

The schema theorem assures that excellent schemas will increase their frequency in the population at an exponential rate, which provides the possibility of GA obtaining the optimal solution. The building bock hypothesis indicates that GA has the capacity to obtain the global solution finally.

(3) *Characteristics of GA*

GA is based on the mechanism of natural selection and population genetics, based on the iterative, evolutionary process, and has a wide range of practical application. GA combines the two mechanisms that survival of the fittest in the biological evolution and random exchange of information. Comparing with traditional optimization algorithms, GA has the characteristics of population strategy and genetic operation. Its characteristics could be summarized as following:

(i) GA uses the encoding strings (chromosomes) as the searching object. Traditional optimization algorithms always use the variables itself as object of operation, however GA utilizing the other form of variables, chromosomes, as the optimization object. The encoding of feasible solution, chromosomes, is easily to achieve the simulation of biological mechanisms in nature and the principle of genetic variation.

(ii) GA researches the feasible solution from population instead of single individual. Traditional optimization algorithms search the optimal solution via iteration from single initial point, which provides the information insufficiently that the algorithm is easily trapped in local solution. GA is from population which covers widely and conducive to global optimization.

(iii) GA needn't basically knowledge of the search space or other ancillary information, but only using the fitness function to evaluate the individual. The fitness function is not only unconstrained, but also its domain can be set arbitrarily. This feature makes the application of GA to being greatly expanded.

(iv) GA is not deterministic rules, instead of using probability rules to guide and change the direction of its search.

(v) Essence of GA is a series of operations on schemas. Every individual actually contains multiple schemas; GA operates on the population, actually operates the schemas more than number of individuals in population. GA actually has implicit parallelism.

(4) *Application of GA*

In management science, operations research and engineering design, optimization problems take a very important position. The complexity of combinatorial optimization problems in many industrial fields is same as NP-hard problem. As the problem scale increases, the solution space of combinatorial optimization problems increases rapidly. To such complex issues, people have

realized that the main focus should be on a satisfactory optimal feasible solution instead of exact solution. GA is the one of the satisfactory algorithms to search this kind of satisfactory optimal feasible solution. Practice has proved that GA for combinatorial optimization problems is very effective. For example, GA has successful application in solving the traveling salesman problem, knapsack problem, bin packing, graph partitioning problem etc.

In addition, GA is also applied successful in production scheduling, automatic control, robotics, image processing, artificial life, genetic coding, and machine learning etc.

2.1.2 Fish Swarm Optimization

(1) *Brief Overview*

In 1990s Dr. Tu Xiaoyuan researched and developed a new computer animation, "artificial fish", which is known by academics as "xiaoyuan's fish" [55]. "xiaoyuan's fish" was cited by the English-speaking countries in general mathematics textbooks, and was introduced extensively by academic journals of western countries.

In 2000s, Dr. Li Xiaolei also proposed a fish swarm optimization (FSO) approach to solve the combinatorial optimization problems [50]. As one of the evolutionary computation, FSO was researched and developed by academics. This thesis focus on behavior operations and operation selection rule, researches, improves and applies it to the stock layout problem.

(2) *Definition of FSO*

In the midst of the waters such as oceans and lakes, fish swarm typically can find nutrient-rich areas quickly. Thus there is most nutrient at the field where a largest number of fishes live in. Based on this characteristic, FSO simulates the process of fish swarm foraging, clustering and following to solve the optimization problems. The following is several typical behaviors of fish swarm:

(i) *Foraging behavior*. In general, fishes swim freely and randomly in the water. When they find food, they will swim quickly to the way where the food is gradually increased

(ii) *Clustering behavior*. During the fish swarm swimming to move, in order to protect its own survival and avoid from harming fishes naturally cluster as a group. In the processing of fishes clustering, they follow three rules:

a. *Separation rule*. They always try to avoid from overcrowding and keep some distance each other.
b. *Alignment rules*. They try to keep the same direction of moving with partners as much as possible.
c. *Cohesion rule*. They try to move toward the center of the partners.

(iii) *Following behavior*. When one or several fish find food, the other fishes near them will swim moving to the way of food found quickly.

The algorithm model of FSO uses the model in the same way as most swarm intelligence algorithms, such as that of the genetic algorithm (GA). That is, first it analyzes the problem and determines the coding method for optimization of individuals (artificial fishes). Next, it generates and initializes the original population of individuals. And then, according to the behavioral characteristics of the fish in the swarm, performs the optimizing iteration of the population using the algorithm operations. Finally, it decodes the best individual obtained as the layout pattern solution. We also give the flowchart of the FSO as Fig.2.2.

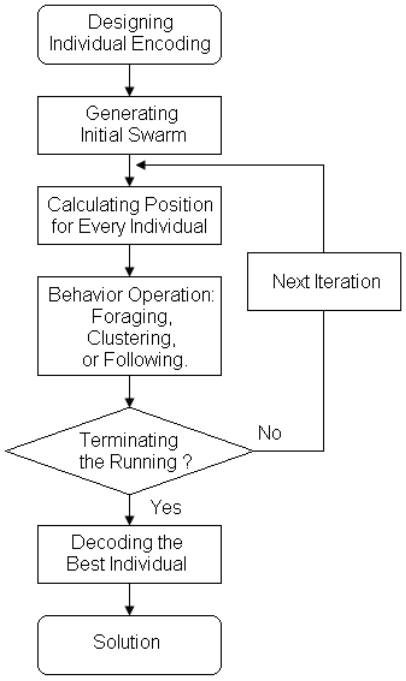

Fig. 2.2 Flowchart of the FSO.

(3) *Characteristics of FSO*

FSO is based on simulating the process of fish swarm foraging, as same as GA, also based on the iterative process. FSO also could be applied at a wide range of practical project optimization. Comparing with traditional optimization algorithms and other evolutionary computations, the characteristic of FSO could be summarized as following:

(i) FSO Has a faster convergence rate; could be applied to solve the problems with the requirement of real-time control.

(ii) To some optimization problems without requiring solution so precision, FSO could obtain a feasible solution

quickly.

(iii) FSO does not require the strict mechanism model of the problems; do not even need the exact description of the problems. This characteristic makes its scope of application can be extended.

There are some cases could be the termination condition during the FSO running.

(i) Mean Square Error (MSE) obtained repeatedly for many times are less than the allowable error.

(ii) The artificial fishes are overcrowding over certain allowable degree.

(iii) The best individual has no any improvement for many repeating iteration.

(4) *Application of FSO*

As one of the evolutionary computation, FSO also could be applied to solve the project problems without requiring the solution with high precision. Such as typical combinatorial optimization problems: traveling salesman problem, knapsack problem, bin packing, graph partitioning problem etc.

In addition, FSO could also be applied in production scheduling, automatic control, robotics, image processing, artificial life, genetic coding, and machine learning etc.

Especially, FSO could be applied in real-time system as its characteristic of fast convergence. To solve the small scale problems, characteristic of fast convergence of FSO is not so obvious comparing with other evolution algorithm. But to large scale problems FSO demonstrates excellent rapid characteristics to obtain the optimal solution.

FSO is still a young algorithm with still some defeats. Although FSO obtains the optimal solution rapidly, its feature of convergence still need be improved, and its computational complexity also should be developed and improved.

2.2 Knapsacks Problems

Combinatorial optimization problems are a class of discrete optimization problems, which have been widely used in programming, scheduling, resource allocation, and decision-making etc. The typical combinatorial optimization problem includes Traveling Salesman Problem (TSP), Scheduling Problem such as Flow-Shop and Job-Shop, Knapsack Problem, Bin Packing Problem, Graph Coloring Problem and Clustering Problem etc. The mathematical descriptions of these problems are very simple, and they have strong engineering representative, but it is difficult to solve the optimal solution.

Knapsack Problems includes many kinds of problems, such as Multi-choice multidimensional Knapsack Problem, Multi-demand multidimensional Knapsack Problem, Multi-Dimensional Knapsack Problem and Multi-Knapsack Problem etc. This section introduces those the thesis involved in with mathematical descriptions.

2.2.1 Multi-Dimensional Knapsack Problem

The Multi-Dimensional Knapsack Problem (MDKP) is an NP-hard problem that has several practical applications, such as processor allocation in a distributed system, cargo loading, stock cutting, project selection, or capital budgeting. The goal of the MDKP is to find a subset of objects that maximizes the total profit while satisfying some resource constraints, which can be formulated as:

$$Maximize \sum_{j=1}^{n} v_j x_j \tag{2.1}$$

$$s.t. \sum_{j=1}^{n} w_{ij} x_j \le c_i, \quad \forall i \in I$$

$$x_j \in \{0,1\}, \quad \forall j \in J \tag{2.2}$$

where n is the number of objects, m is the number of resources, v_j is the value associated with object j, w_{ij} is the consumption of resource i for object j, c_i is the available quantity of resource i (capacity of knapsacks for the i^{th} resource), and x_j is the decision variable with object j and is set to 1 if j^{th} object is selected (and is otherwise set to 0). $I = \{ 1, 2, \dots, m \}$, $J = \{ 1, 2, \dots, n \}$.

Constraints c_i ($\forall i \in I$) described in Eq.(2.2) are referred to as knapsack constraints, so the MDKP is referred to as the m-dimensional knapsack problem. A number of authors also include the term zero-one when referring to the problem, e.g., the multi-dimensional zero-one knapsack problem. To discriminate Multi-Knapsacks Problem described in next section, this thesis calls the Multi-Dimensional Knapsack Problem for short as MDKP.

Most of researches on MDKP are involved genetic algorithm and other evolutional algorithm. P.C. Chu (1998) presented a genetic algorithm for MDKP with a repair operator [7]. Günther R. Raidl (1999) proposed a Weight-Coding in a GA for the MDKP [8], in this algorithm, two type of heuristic function combining 4 commutations of v_j are designed as a set of algorithm for MDKP. Jens Gottlieb (2000) solved the MDKP with a Permutation-Based GA [15], which encoded the chromosome, performed the crossover operator and mutation operator entirely based on a permutation of objects. Farhad Djannaty (2008) proposed the GA focus on penalty function for MDKP [9]. Günther R. Raidl

(1998) presented an improved hybrid GA for MDKP [11]. Alex S. Fukunaga (2008) designed the GA for MDKP which searches a space of undominated candidate solutions [12]. Alex S. Fukunaga (2009) combined several representations in GA for MDKP [13].

At fields of other evolutional algorithm, V.Gabrel (2002) separated the MDKP as sub-problems to solve [16]. In this algorithm an exact separation scheme is presented for identifying most violated extended cover inequalities for application to MDKP. The minimality of the resulting covers is shown to be a basic property of the criterion used for separation. Maoguo Gong (2007) introduced a computational model simulating the dynamic process of human immune response to solve MDKP [17].

2.2.2 Multi-Knapsack Problem

Multi-Knapsacks Problem (MKP) is a classical NP-complete combinatorial optimization problem with applications in various fields such as processor allocation in distributed system, cargo loading, cutting stuck, project selection, or capital budgeting. The problem is to identify a subset of objects that maximizes the total profit while satisfying some resource constraints. More formally, a MKP is stated as follows:

$$Maximize \sum_{j=1}^{n}\sum_{i=1}^{m} v_j x_{ij} \tag{2.3}$$

$$s.t. \sum_{j=1}^{n} w_j x_{ij} \le c_i,\ i \in I \tag{2.4}$$

$$x_{ij} \in \{0,1\}, \sum_{i=1}^{m} x_{ij} \le 1,\ i \in I,\ j \in J \tag{2.5}$$

where m is the number of knapsacks, n is the number of objects. v_j is the value associated with object j and w_j is the weight of object j, x_{ij} is the decision variable which is set to 1 in case that object j is in knapsack i, otherwise to 0. c_i is capacity of knapsack i, $I = \{ 1, 2, \dots , m \}$, $J = \{ 1, 2, \dots , n \}$.

In the literatures about knapsack problems, the Multi-Dimensional Knapsack Problem is called for short as MKP most often. To discriminate these two knapsack problems, the thesis call the Multi-Knapsacks Problem for short as MKP.

2.3 Stock Layout Problem with Guillotine Cutting

Stock layout problem exist in many industrial fields, including shipbuilding, automobile, glass cutting, and furniture production industries. These problems are most typically with respect to a rectangular shape—that is, how to reasonably cut large rectangular sheets (stocks) into the necessary small rectangular parts (items) in order to economize the material of the sheets. Thus, the goal of such a problem is to use the least number of stocks to produce the required number of items. This type of problem is also called a stock layout problem.

2.3.1 Stock Layout Problem

There is a set of available stocks A consisting of A_i with length L_i, width W_i, and texture direction T_i (either lengthwise or widthwise), i=1, 2,…,m. We assume there is sufficient A. There is also a set of demanded rectangular items B consisting of B_j with length l_j, width w_j, and texture direction t_j (either lengthwise or widthwise), j=1, 2…n.

The problem is to choose a subset A' from A, and a layout pattern cutting A' into B to maximize the objective function:

$$Utilization = \left(\frac{\sum_{j=1}^{n} l_j w_j}{\sum_{i=1}^{m_c} L_i W_i - R} \right) \tag{2.6}$$

where m_c (< m) is the number of stocks in A', and in general the first $m_c - 1$ stocks of A' are used in full, and the last stock is partially used. R is the residual area of the last stock chosen in the layout pattern and appears with a minus sign because area R is not considered waste when all of items have been finished. (The remainder with area R can be used for another project.)

As a combinatorial optimization problem, stock layout problem has the constraints as follows:

(1) Don't allow to overlap each other between B_i and B_j, $i, j = 1, 2,\dots,n$, $i \neq j$.
(2) B_j (j = 1, 2,…,n) locate within rectangular stocks.
(3) Items are placed according the same texture with stocks.
(4) Satisfying the requirement of guillotine cutting.

Refs. [40]-[42] present algorithms for the stock layout problems, but only including requirement (1). Refs. [43]-[44] present algorithms which include both requirements (1) and (5). Andreas Bortfeldt considers the problem under requirements (1), (2), and (4) [45]. Some studies define the problem and the optimization goal differently [46]-[48]. Ref. [49] presents research focused particularly on number of guillotine partitions in d dimensions. However, there are no algorithms that work under all of the cases that we have described as arising in practice production.

2.3.2 Guillotine Cutting in Stock Layout Problem

In practical production processes, plate work for cutting stocks is often done by machinery in such a way that cutting the sheet metal stocks often requires a layout pattern consistent with guillotine cutting. This requirement arises because the cutting process is almost impossible to stop at a precise point internal to the stock. This makes it difficult to avoid short-cutting or over-cutting, where the former makes it difficult to separate the parts, the latter leads to damaging other sub-pieces.

Guillotine cutting is defined as cutting such that every cutting operation divides the stock into two pieces; in particular, the cutting line runs from one edge of a rectangle to the opposite edge, parallel to the other two edges. Thus, the guillotine-cutting requirement is an additional constraint on the general stock layout problems.

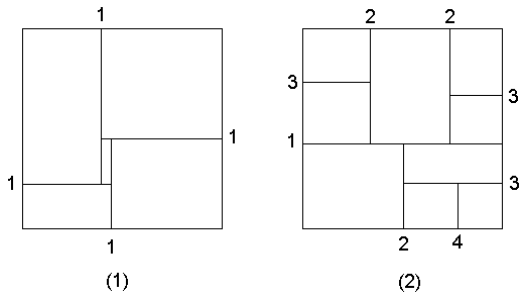

Fig. 2.3 (1) It is hard to cut away the piece by machine processing.(2), It is complete to cut away all of sub-pieces from one-piece just in a certain turn of cut lines.

As Fig.2.3(1), it isn't meeting the guillotine cutting, because that cutting beginning from any line is hard to avoid from wounding the opposite piece of cutting line. Layout like Fig.2.3(2) make it easy to cut away each sub-pieces from stock piece just in turn that cutting along line 1 firstly then line 2, 3 and 4 as meeting guillotine cutting.

2.4 Summary of the Combinatorial Optimization Problems

We give the summary of the combinatorial optimization problems in operation research problems for reader more convenient find it.

(1) Bin Packing Problems:
- One-dimensional
- Two-constraint
- Two-dimensional

(2) Knapsack Problems:
- Multi-choice multidimensional
- Multi-demand multidimensional
- Multidimensional
- Multiple
- Quadratic
- Sharing

(3) Location Problems:
- Capacitated vertex p-centre
- Capacitated warehouse location
- Capacitated warehouse location, single source
- P-hub
- P-median - uncapacitated
- P-median - capacitated
- Uncapacitated warehouse location
- Weber (continous) location

(4) Network Flow Problems:
- Multicommodity
- Single commodity
- Single commodity, concave costs, single source, uncapacitated

(5) Scheduling Problems:
- Aircraft landing
- Common due date
- Flow shop
- Hybrid reentrant shop
- Job shop
- Lot streaming
- Multiprocessor task scheduling in multistage hybrid flowshops
- Open shop
- Shift minimization personnel task
- Weighted tardiness
- Weighted tardiness with sequence-dependent setup
- Workforce

(6) Shortest Path Problems:
- Multiple objectives
- Resource constrained

(7) Steiner Problems:
- Euclidean Steiner problem
- Prize collecting Steiner problem
- Rectilinear Steiner problem
- Steiner problem in graphs

(8) Three-Dimensional Cutting/Packing Problems:
- Boxes on shelves
- Container loading
- Container loading with weight restrictions

(9) Travelling Salesman Problem:
- Multiple objective salesman

Period salesman
Single period

(10) Two-Dimensional Cutting/Packing Problems:
Assortment problem
Constrained guillotine (Stock layout)
Constrained non-guillotine
Non-rectangular items
Strip packing
Unconstrained guillotine

(11) Vehicle Routing Problems:
Dial-a-ride
Fixed areas
Fixed routes
Inventory routing
Multi-depot
Multi-depot with time windows
Period routing
Period routing with time windows
Single period
Single period with pick-ups and deliveries
Single period with time windows
Single period with time windows and pick-ups and deliveries
Site-dependent routing
Site-dependent multi-trip period routing
Site-dependent with time windows
Sparse feasibility graph
Two-echelon
Urban transit

Chapter 3

A Genetic Algorithm for the Multi-Dimensional Knapsack Problems

Solving combinatorial optimization problems with genetic algorithms has attracted a great deal of attention over the past two decades. As combinatorial optimization problems have constraints condition, a number of lethal chromosomes which bring out the infeasible solutions are often produced in the chromosome pool. The lethal chromosomes contain evolutionary achievements with an important feature, which can contribute to the evolution of the genetic algorithms. In the present study we discover this important feature of lethal chromosomes; and propose an approach in genetic algorithms using the lethal chromosomes based on immune operations for the multi-dimensional knapsack problems. From examination it could be concluded that using the lethal chromosomes can improve effectively the performance of genetic algorithms.

3.1 Lethal Chromosomes in Genetic Algorithm

Over the last two decades, as a result of high global search performance and robust performance, genetic algorithms (GA) have been widely applied to large-scale combinatorial optimization problems [5]-[13]. With representing the solution of problem as a chromosome, GA searches the feasible chromosome that satisfy the constraint conditions with the objective function over the entire genetic space. The chromosomes that violate the constraint conditions are referred to as lethal chromosomes (LCs).

In the population of GA, due to crossover and mutation operations, LCs are often generated with high rates, the more LCs may be produced by genetic operations in the search process to approach to the optimal solution, especially in combinatorial optimization problems having severe constraints. The greater number of LCs in the population, the worse search performance of the GA, in the worst case, the algorithm even ceases to run. After population is evolved for several generations, the LCs generated contain the excellent feature despite they are infeasible. Abandoning the LCs from population equals to waste the useful resources of evolution.

Research focusing on the problems associated with LCs remains rare. Iima Hitoshi (1995) investigated the effects of LCs on the performance of the GA but did not propose a method for handling these problems [5]. Mengchun Xie (1996) proposed an algorithm model called the double islands model to revive the LCs by random crossover and mutation operations [6]. Due to its randomness, and without using characteristic information, the efficiency of the double islands model algorithm must be improved.

P.C. Chu and J.E. Beasley (1998) introduced a genetic algorithm for MDKP [7], which handled the LCs with a repair operator. In this repair operator, the LCs were revived based on a definition of pseudo-utility ratio. Farhad Djannaty gave some penalty functions for infeasible solution [9]. The penalty function is the most common approach in the GA community to handle the constraints.

In any GA implementation for constrained optimization problems, it is an important issue to handle constraints. A number of procedures were described which considered the constraint in an optimization problem. Zbigniew Michalewicz [14] (1996) presented a suitable classification of these procedures which are described (i) rejecting strategy, (ii) repairing strategy, (iii) modification of genetic operators, (iv) penalizing strategy.

Table 3.1 Notations and functions used in this chapter

Notation	Meaning
m, n	The number of resources and objects
I, J	$I = \{ 1, 2, \dots, m \}, J = \{ 1, 2, \dots, n \}$
v_j	The value associated with object j
w_{ij}	The consumption of resource i for object j
c_i	The available quantity of resource i
x_j	The decision variable with object j be selected or not
$x_1x_2\dots x_n$	The chromosome associated with a solution of the problem
$fitns(x_1x_2\dots x_n)$	Return the fitness of a chromosome
$s_1s_2\dots s_n$	The vaccine schema
t	The threshold be used to binary-value process the $s_1s_2\dots s_n$

In the present chapter, we use the repairing strategy to propose an immune genetic algorithm (IGA) for the multi-dimensional knapsack problems (MDKP), which handles the LCs with an immune operation.

3.2 An Immune Genetic Algorithm for the MDKP

Above all, the notations and functions to be used often in the text firstly are listed in Table 3.1. They will be explained at first use in the text. However, the reader may find it more convenient to look up definitions in the table.

3.2.1 Algorithm Model

The proposed IGA has two types of chromosome pools, namely, the living island and lethal island. The former contains chromosomes, referred to as non-lethal (feasible) chromosomes which satisfy all constraints. The latter consists of the LCs. In the living island, chromosomes are evolved by genetic operations, and in the lethal island, LCs are revived by immune operations.

In the IGA model, after initializing the population, the population is divided into two islands according to whether the chromosome is lethal or non-lethal. A flowchart of the double islands model is shown in Fig.3.1.

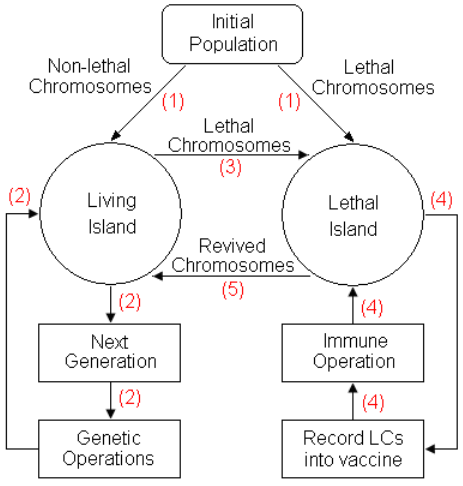

Fig. 3.1 Flowchart of the double islands model.

With generating the initial population firstly, the IGA work under a repeat process that: performing the genetic operations at living island, moving the LCs from living island to lethal island, performing the immune operator for all LCs at lethal island, moving the non-lethal chromosomes from lethal island to living island, and then to performance genetic operations again at living island. In Fig.3.1 this process is described by a loop of flow numbered as: (2)(3)(4)(5)(2)… . The steps of IGA are summarized as following:

(1). *Initialize population*

Evaluate fitness of chromosomes in the population. Move the non-lethal chromosomes into the living island and the lethal chromosomes into the lethal island.

(2). *Evolve population*

(In the living island):

All of the chromosomes in the living island evolve into the next generation by genetic operations (selection, crossover, mutation), and the new lethal chromosomes move to lethal island. In this process, the vaccine described later must be trained.

(In the lethal island):

Each chromosome in the lethal island is handled by immune operation and is then moved to the living island.

(3). *Repeat step (2) until the termination condition of the GA is satisfied.*

IGA mainly works under structure of GA with an immune operator handling the LCs. So we explain the IGA through classifying roughly the IGA into two parts, refer as genetic operations which happen at living island, and immune operation which happen at lethal island. The details of them are explained separately in next sections.

3.2.2 Genetic Operations

Genetic algorithm is a swarm intelligent algorithm which works on the Darwin's principle of natural selection. It is well–known to researchers who work on optimization method. So we don't repeat to introduce the GA in detail, just to state the operations which are involved in our IGA. Standard genetic operation includes representation, selection, crossover and mutation, so we state them below.

(1) *Representation*

The standard GA 0-1 binary representation is adopted in our IGA, which has been proven to be well suited for different combinatorial optimization problems. To MDKP, the chromosome is a n-bit binary string $x_1x_2\ldots x_n$, where n is the number of objects. In this representation a value of 0 or 1 at the j^{th} bit implies that $x_j = 0$ or 1 in the solution. A fitness of chromosome is obtained by:

$$fitns(x_1x_2\ldots x_n) = \begin{cases} \sum_{j=1}^{n} v_j x_j, \sum_{j=1}^{n} w_{ij} x_j \le c_i \ (\forall i \in I) \\ 0, \quad \sum_{j=1}^{n} w_{ij} x_j > c_i \ (any\, i \in I) \end{cases} \tag{3.1}$$

By the way, with representation in order to facilitate the description, we present several definitions about

chromosome. A *chromosome* is denoted by a binary value string composed of *n* bits, the bit of chromosome is referred to as a *gene*. Some genes can make up an incomplete chromosome, which is a combination of several genes in a chromosome. We refer to an incomplete chromosome as a *block*; the number of genes in a block is referred as *block length*.

(2) *Selection, crossover and mutation*

Parent selection is to assign the reproductive opportunities to each individual in the population, and select two as parent who will have children. Roulette wheel selection is adopted in this chapter, which is based on fitness of individual to assign the reproductive opportunities for each individual. With selecting two individual as parent, perform the crossover operator to get the two children.

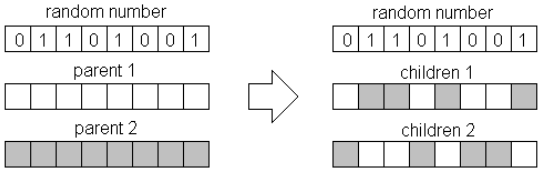

Fig. 3.2 Crossover operator

Crossover operator is to exchange some genes between two parent chromosomes and then get two new chromosomes as children. From first gene to last gene, crossover operator decides to exchange the two values of genes or not with *n* 0-1 random number. If random number is 1 to exchange the value of gene otherwise don't exchange. The crossover manner for an example $n = 8$ is described in Fig.3.2.

Once two children chromosomes have been generated through crossover operator, a mutation operator is performed that mutates several randomly selected genes in the children chromosomes. These selected genes are changed from 1 to 0 or vice versa. Generally, the rate of mutation is set to be a small value, which is set as 0.05 in IGA.

3.2.3 Immune Operation

(1) *Idea of immune operation*

Immune idea was introduced from biology and medicine. Long time ago, people noted that patients with infectious diseases were healed; the immunity to the disease was generated in their body. Immune operation of IGA just works based on this phenomenon. Although, there are several difference definitions for artificial immune system currently [17]-[19], such as simulation of vaccine and vaccination, antibody and antigen of immune system, and clone selection principle etc, anyway the most classic immune system as Fig.3.3. The previous infection led the system to generate the immune memory; the system handles the re-infection with previous memory as vaccine for avoiding from being infected again. The characteristic of the immune system is a memory effect.

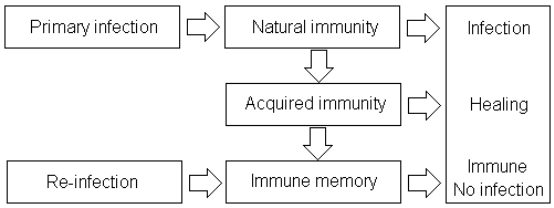

Fig. 3.3 Immune system

In this chapter about GA for MDKP, the process that chromosomes become to LCs as crossover and mutation likes an infection of immune system. We record the statistical information of previous LCs as immune memory, and then handle the succeeding LCs by previous memory. Accordingly we try to simulate the immune system process in such a double-islands GA. The operation to handle the LCs is called immune operation in this chapter.

(2) *Immune operations in IGA*

In order to record the statistical information of LCs appeared, a multi-valued string schema $s_1s_2\ldots s_n$ is constructed and has the initial value that $s_j = 0$, $\forall j \in J$. The $s_1s_2\ldots s_n$ will be used as *immune memory* by immune operation, so we call it *vaccine*. During the IGA at the stage of evolution, every LC $x_1x_2\ldots x_n$ generated by genetic operations is recorded into $s_1s_2\ldots s_n$ by following way.

(Algorithm: record the LCs into vaccine)
for j=1 *to n do*
  if x_j=1 ***then***
    $s_j \leftarrow s_j+1$;
  end if
end for

By this way the $s_1s_2\ldots s_n$ is a changing string, which is refurbished by every lethal chromosome while evolution of population as long as LCs appearing. As LCs generated are handled by immune operation in every generation, in the other word, while the vaccine is refurbished by LCs while the vaccine is used by immune operation in IGA.

Our immune operation consists of two phases. The first phase is to compare the LC to be handled with best chromosome of population for every gene, mark the some genes which have the same value as corresponding genes of best chromosome, and keep them without changing in

the next phase. Those genes make up of a *block*, this phase is to protect such an excellent block, we named the phase as PEB, those same value genes are *protective genes* and others are *non-protective genes*. The purpose of the PEB is to extract a *block*, which is regarded as evolutionary achievement, close to the optimal chromosome and should be protected. As the exact chromosome is unknown and is tentatively substituted by the best chromosome in phase PEB.

The second phase includes two steps. The step.1 examines each *non-protective gene* in *increasing* order of s_j's and change the gene from one to zero if LC is still lethal. The step.2 reverses the process by examining each *non-protective gene* in *decreasing* order of s_j's and changes the gene from zero to one as long as the chromosome is not a LC. Since this phase is based on vaccine $s_1s_2...s_n$, which records the information of previous LCs, it could be called *vaccination* in immune operation. The step.1 is to obtain a non-lethal chromosome from LC, whilst step.2 is to improve its fitness further.

In order to achieve an efficient implementation of the immune operation, a preprocessing routine is applied to vaccine $s_1s_2...s_n$ that sorts and renumbers genes of LC and best chromosome according to the *increasing* order of s_j's. Let $x_1x_2...x_n$ is the preprocessed LC to be handled, $b_1b_2...b_n$ *is* preprocessed best chromosome. Proposed immune operation is described by pseudo-code as following.

(Algorithm: immune operation)
Let $W_i = \sum_{j=1}^{n} w_{ij}x_j$, $\forall i \in I$; $p_j = 0$, $\forall j \in J$
for j=1 *to* n ***do*** *//PEB phase*
  if (x_j=b_j) ***then***
    $p_j \leftarrow 1$; *// p_j=1 implies x_j is protective gene*
  end if
end for
for j=1 *to* n ***do*** *//step.1 of vaccination phase*
  if (x_j=1, p_j=0) *and* ($W_i > c_i$, *for any* $i \in I$) ***then***
    $x_j \leftarrow 0$;
    $W_i \leftarrow W_i - w_{ij}$, $\forall i \in I$;
  end if
end for
for j=n *to* 1 ***do*** *//step.2 of vaccination phase*
  if (x_j=0, p_j=0) *and* ($W_i + w_{ij} \le c_i$, $\forall i \in I$) ***then***
    $x_j \leftarrow 1$;
    $W_i \leftarrow W_i + w_{ij}$, $\forall i \in I$;
  end if
end for

Immune operation marks the genes which are same as the corresponding genes of best chromosome in PEB phase (first *for loop*). The step.1 of vaccination (second *for loop*) is to remove some genes as zero excepted protective genes until a feasible chromosome is achieved. The step.2 (third *for loop*) adds the genes as one whilst preserve a feasible chromosome.

3.3 Computational Experiments

3.3.1 Feature of Lethal Chromosomes

In this section we analysis the feature LCs contain, the LCs are recorded into vaccine $s_1s_2...s_n$ in the process of evolution, so we study vaccine here. Above all we firstly give a definition of *similarity ratio* about two chromosomes. There are two chromosomes X_1 ($x_1^1x_2^1...x_n^1$) and X_2 ($x_1^2x_2^2...x_n^2$), the *similarity ratio* of X_1 and X_2 is defined by:

$$similarityratio(X_1, X_2) = 100\frac{1}{n}\sum_{j=1}^{n}(x_j^1 \otimes x_j^2) \quad (3.2)$$

where, if $x_j^1 = x_j^2$, then $x_j^1 \otimes x_j^2 = 1$; otherwise $x_j^1 \otimes x_j^2 = 0$. Actually, the *similarity ratio* is used to measure how much the two chromosomes have same value genes.

As described in section 2.4, the vaccine $s_1s_2...s_n$ is a multi-valued string that s_j ($\forall j \in J$) is the accumulated value of x_j of LCs generated. Vaccine is a changing string with generation advancing as long as LCs are generated in population. In order to study feature the vaccine contained, It is hypothesis at a certain generation that we make binary-value processing to $s_1s_2...s_n$ and then get a binary-value string $s'_1s'_2...s'_n$ by the way that:

for $\forall j \in J$, *if* ($s_j > t$) *then* $s'_j \leftarrow 1$, *otherwise* $s'_j \leftarrow 0$.

where t is the threshold be used to classify the s_j's into one or zero, the t is determined as an appropriate value so that it takes the max *similarity ratio* for $s'_1s'_2...s'_n$ and *exact chromosome* (exact solution) of the problem. Here it is assume that the exact chromosome is known beforehand. We call the value of *similarity ratio* of $s'_1s'_2...s'_n$ and exact chromosome of problem as *feature of vaccine*. Since vaccine is a changing string with generation advancing as long as LCs are generated, the *feature of vaccine* is also a changing value with generation advancing. In this section we test the *feature of vaccine* how to change for against to generation on two MDKP instances.

We picked up two MDKP instances that m = 5, n = 100 from OR-Library to test on. OR-Library can be referenced at http://people.brunel.ac.uk/~mastjjb/jeb/info.html [7]. To this two MDKP instances, we firstly solve their exact chromosomes with branch and bound method (BBM) so

as to calculate the *feature* of vaccine. The curves of *feature of vaccine* for two MDKP instance are shown as Fig.3.4(1) and (2) separately.

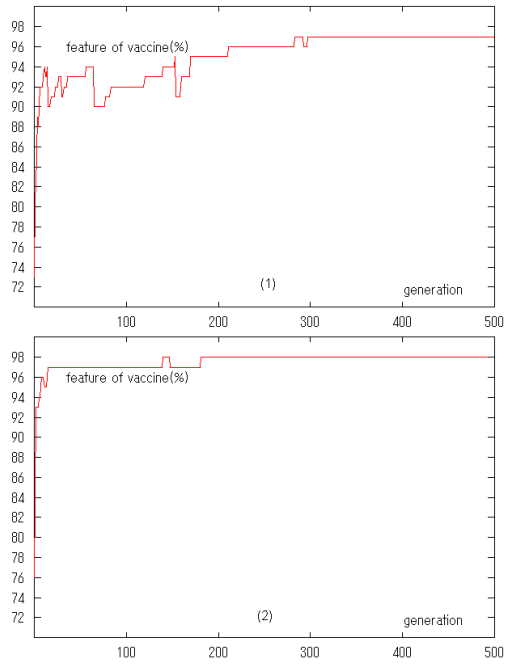

Fig. 3.4 Feature of vaccine ["generations" (×2), "feature of vaccine" (×2).]

We could learn from Fig.3.4 that (i) the *features of vaccine* are increase in the overall; (ii) the *features of vaccine* raise to a high degree at later generation that Fig.3.4(1) to 97 % and Fig.3.4(2) to 98 %. That also means most of genes of exact chromosome could be indicated the value via to classifying the corresponding *bit* of vaccine with a threshold *t*. Although the *t* is unknown, according to proposed algorithm descried in 3.4, immune operation select the smaller value s_j's to set corresponding genes of LCs from one to zero and select greater value s_j's to set genes from zero to one. The median value s_j's are used not so often. By this way that the immune operation based on *feature of vaccine* could guides several genes of LCs to become the value of corresponding genes of exact (optimal) chromosome effectively.

If only two instances cannot enough to show the *feature of vaccine* for MDKP, we also have test *feature of vaccine* on a set of small scale MDKP picked from OR-Library, which consists of 55 MDKP instances that m = 2 to 30 and n = 6 to 15. To each problem the IGA stop the running when exact chromosome is found, then obtain the finial *feature of vaccine* and write down the finial generation. Averagely to 55 problems, the exact chromosomes are obtained at 40^{th} generation (population size is 50) and finial average of *feature of vaccine* is 87.9 %. That is, to 55 small problems IGA obtain the exact solution at case of that *feature of vaccine* is 87.9 % averagely but finial generation is only 40, the finial generation is only 40 so that the finial *feature of vaccine* is not enough high. If IGA is used to solve lager scale MDKP with serious constraints, the LCs are generated more often, the *feature of vaccine* will rises more quickly, the effects of immune operation based on vaccine is self-evident.

Although vaccine added up from LCs shows so amazing characteristics that it could be binary-value processed as a chromosome which is very close to optimal chromosome, we note that this feature of vaccine must be based on two conditions, (i) the LCs have to be enough to accumulate the vaccine, (ii) the parents to generate the LCs are based on selection by method of survival of the fittest such as roulette method. The two conditions are very important to vaccine, if it is not satisfied the vaccine will not hold the feature like the Fig.3.4.

3.3.2 Testing PEB and Vaccination

Immune operation includes two phases: PEB and the vaccination. In order to test the effectiveness of two phases, IGA is tested at two cases: IGA-1 does not include the PEB but only vaccination in immune operation; IGA-2 includes both of PEB and vaccination. The proposed IGA in this chapter includes both of two phases; we just split it as IGA-1 and IGA-2 here for separately testing. In addition, we also coded the program and test for standard genetic algorithm (SGA), which works without using lethal chromosomes, and a GA with repair operation proposed by P.C. Chu [7]. As P.C. Chu used a repair operation to handle the LCs based on u_j's ($u_j = v_j / \sum_{i=1}^{m} w_{ij}/c_i$, $j \in I$), we call his algorithm as *RGA*.

55 standard test problems are available from OR-Library. These problems are divided into six different groups and are real-world problems consisting of m = 2 to 30 and n = 6 to 15. This set of test problems have been also used by other researchers [7] [9]. We solved these problems on our computer (Celeron 1.0) with SGA, RGA, IGA-1 and IGA-2 which were coded in Visual C++.net (2003).

Because mentioned algorithm all obtained the exact solution in this experiment, we report only the CPU time which is spent until the optimal is obtained as Table 3.2. The first two columns in Table 3.2 indicate the problem group name and the number of problems in that group. The next four columns separately report the average of exact solution time (CPU seconds) for every problem group to SGA, RGA, IGA-1 and then IGA2. The last row

of Table 3.2 reports the average of CPU time, which is not the average of six sub-average data but the 55 problems all.

In this section we temporarily verify the effectiveness of SGA, RGA, PEB and vaccination separately by comparing them each other with our own procedures. We will test the overall capacity of IGA on large scale problems at next section.

As Table 3.2, IGA-2 is most quick at 3 groups out of 6 groups. The IGA-1 is also most quick at other 3 groups. But overall average data reported at last row shows that the speed of algorithm as an order IGA-2, IGA-1, RGA and then SGA. The RAG, IGA-1 and IGA-2 are all quicker than SGA indicates that LCs are worth to use rather than be abandoned in GA that SGA does not use the LCs. IGA-1 is quicker than RGA indicates that s_j's is more effective as vaccine than u_j's ($u_j = v_j / \sum_{i=1}^{m} w_{ij}/c_i$, $j \in I$). IGA-2 is quicker than IGA-1 shows that phase PEB is necessary in immune operation.

Also, we give the overall evolutional curves of above mentioned four algorithms in Fig.3.5. For each algorithm we give the overall average curve over 55 problems. As 55 problems have very different optimal solutions, we poise the weight for every problem in overall average curve. So in Fig.3.5, the ordinate is the average of 55 problems for 100×(*fitns*/*optimal*), where *fitns* is the fitness of best chromosome in population, and *optimal* is the optimal solution value. The abscissa is the CPU time millisecond (*ms*). It could be learn from Fig.3.5 that the curve of IGA-2 rise to 100 percent before 1000 millisecond, that is IGA-2 obtained the optimal solutions for all 55 problems within one second of CPU time; IGA-1 is second quicker one and the SGA is most poor one according to their evolutional performance.

Table 3.2 Computational results for exact solution time (CPU seconds)

Problem set name	No. of problem	SGA	RGA	IGA-1	IGA-2
HP	2	0.367	0.087	0.140	0.086
PB	6	0.677	0.388	0.187	0.169
PETERSEN	7	0.254	0.312	0.003	0.234
SENTO	2	2.179	0.494	0.273	0.617
WEING	8	0.437	0.227	0.015	0.089
WEISH	30	0.462	0.166	0.192	0.082
Average		0.516	0.249	0.153	0.132

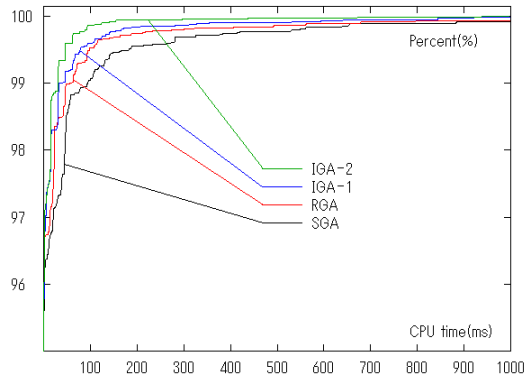

Fig. 3.5 Overall evolutional curves over 55 problems

3.3.3 Testing IGA on Large Scale Problems

Another set of standard test data of the MDKP was provided also by OR-Library which was presented by J.E. Beasley, referred and used by many researchers such as P.C. Chu [7], Günther R. Raidl [8] [11], Jens Gottlieb [14] and V. Gabrel [16]. These test data contain10 problem instances for each combination of $m \in \{5, 10, 30\}$, $n \in \{100, 250, 500\}$, and $\alpha \in \{0.25, 0.50, 0.75\}$ with $\alpha = c_i / \Sigma_{j=1}^{n} w_{ij}$ being the tightness ratio of instance. Since the exact solution values for most of these problems are not known, the quality of a solution is measured by the percentage gap of the objective value *fitns* with respect to the optimal value of the LP-relaxed problem f^{LP}_{max}: *%-gap* = $100\times(f^{LP}_{max} - fitns) / f^{LP}_{max}$. The proposed IGA is tested on these MDKP instances and the results are shown in Table 3.3.

We also list the available results of other references in Table 3.3 to compare with. The first three columns in Table 3.3 indicate the sizes (m and n) and the tightness ratio (α) of a particular problem structure, with each problem structure containing 10 problem instances. The next columns in turn report the other results of average *%-gap*. That are *RGA* proposed by P.C. Chu [7], *GA with H1* and *GA with H2* proposed by Günther R. Raidl [8], *Swap* and *Insert* are from Jens Gottlieb [15], and the *Improved GA* reported by also Günther R. Raidl [11] for *%-gap* obtained from initial population to $(10^6)^{th}$ generations. The last column reports the results of our IGA for average *%-gap* obtained with computer condition described in 3.2.

To compare with results, above all we discuss the condition of experiment firstly. The results of *RGA* are under that 10^6 non-duplicate children had been generated. It is also that there are at least 10^4 generations if population size is 100. *GA with H1* and *GA with H2* was tested with population size of 100 and 10^5 solutions had been evaluated. *Swap* and *Insert* had evaluated 10^6 non-duplicate solutions and then get results. *Improved GA* recorded the best solution from initial population to $(10^6)^{th}$ generations. Comparing with them, IGA obtain the results with population size of 50 and running terminates at

$(10^4)^{th}$ generation.

Averagely for all instances, IGA obtained the *%-gap* as 0.5472, which is smaller than the results from the algorithm *GA with H1*, *GA with H2*, *swap* and *Insert*, but except the *RGA* and *Improved GA*. However please allow us to discuss the results of *RGA* and *Improved* in particular below.

P.C. Chu gave the average *%-gap* of *RGA* as 0.54 with two significant figures after the decimal point [7]. Maybe he gave the results with rounding, but anyway with our analysis there are two reasons can indicate that his detailed result of *%-gap* is more than 0.54. The first is that the average of 27 sub-average data he listed is more than 0.54. Secondly, in 27 sub-average data he given, there are at least 12 data that they are absolutely smaller than that of best solution value which were obtained by various algorithms by so far. In other word, his rounding improves the results for sub-average data which decide the overall average result; the rounding also is adopted in overall average value which improves the results again. So the real overall average result of *RGA* with more significant figures is actually not so small which is more than 0.54.

Improved GA of Günther R. Raidl [11] obtained the best average *%-gap* as 0.534 at $(10^6)^{th}$ generation. Because

Table 3.3 Computational results of other algorithms and IGA

Problems			Average %-gap						
m	n	α	RGA	GA with H1	GA with H2	Swap	Insert	Improved GA	IGA (proposed)
5	100	0.25	0.99	1.007	0.989				0.989
		0.50	0.45	0.453	0.455				0.451
		0.75	0.32	0.319	0.318				0.318
	Average		0.59	0.593	0.587	0.586	0.587	1.38-0.59	0.586
5	250	0.25	0.23	0.256	0.273				0.228
		0.50	0.12	0.127	0.132				0.121
		0.75	0.08	0.080	0.087				0.087
	Average		0.14	0.154	0.164	0.169	0.166	0.48-0.15	0.145
5	500	0.25	0.09	0.115	0.126				0.093
		0.50	0.04	0.053	0.057				0.047
		0.75	0.03	0.032	0.037				0.032
	Average		0.05	0.067	0.073	0.116	0.099	0.13-0.04	0.057
10	100	0.25	1.56	1.624	1.707				1.562
		0.50	0.79	0.803	0.827				0.795
		0.75	0.48	0.493	0.519				0.482
	Average		0.94	0.973	1.018	0.986	0.967	1.27-0.95	0.956
10	250	0.25	0.51	0.589	0.664				0.511
		0.50	0.25	0.276	0.311				0.254
		0.75	0.15	0.161	0.188				0.161
	Average		0.30	0.342	0.388	0.398	0.382	0.71-0.29	0.309
10	500	0.25	0.24	0.332	0.385				0.241
		0.50	0.11	0.150	0.196				0.133
		0.75	0.07	0.085	0.126				0.081
	Average		0.14	0.189	0.236	0.315	0.267	0.26-0.11	0.151
30	100	0.25	2.91	3.067	3.075				2.913
		0.50	1.34	1.376	1.478				1.341
		0.75	0.83	0.848	0.942				0.829
	Average		1.69	1.764	1.832	1.748	1.741	3.63-1.71	1.694
30	250	0.25	1.19	1.382	1.615				1.191
		0.50	0.53	0.609	0.706				0.533
		0.75	0.31	0.348	0.449				0.324
	Average		0.68	0.780	0.923	0.821	0.852	1.58-0.64	0.683
30	500	0.25	0.61	0.785	0.995				0.614
		0.50	0.26	0.336	0.447				0.265
		0.75	0.17	0.195	0.331				0.179
	Average		0.35	0.439	0.591	0.631	0.605	0.70-0.33	0.353
	Average		0.54	0.589	0.646	0.641	0.629	1.13-0.53	0.5472

such an experiments to 10^6 generations will cost too CPU seconds to finish for us, we test the IGA for 10^4 generations and obtained the finial average *%-gap* as 0.5472 with population size 50. Relatively, the population size Günther R. Raidl adopted in *Improved GA* is 100. To compare with *Improved GA* at same generations, we calculated the average *%-gap* obtained also at $(10^4)^{th}$ generation for *Improved GA* according to data in Refs.[11], the result is 0.6211 which greater than 0.5472 despite of large population size 100.

3.4 Discussion

3.4.1 PEB and Vaccination in Immune Operation

In a population of GA there is at least one best chromosome no matter what we have already known it or not. If all the chromosomes are sorted as an *increasing order* according to the *similarity ratio* of present chromosome and best chromosome, which are measured by definition of *similarity ratio* described as 3.1, the order obtained is no doubt different from another order that the chromosomes are sorted by *increasing order* according to their fitness. That is also each chromosome has two ways to close to the best chromosome that by *similarity ratio* and by fitness. PEB in immune operation is to maintenance the LCs be changed genes without removing the *similarity ratio* from best chromosome. If some different chromosomes to compare with a same best chromosome in PEB phase, their *protective genes* are different. After being handled by immune operation their genes changed are also different, by this way the original diversity of population is protected so that population can avoid from premature. Of course the best chromosome obtained in the population maybe is not the optimal or exact chromosome, but it is the nearest one from optimal chromosome measured by fitness.

In the past, we have tried to extract the excellent block from LCs with an estimated formula, but the defect is that it costs the CPU time too much, and also the number of the genes should to be extracted is uncertain. Comparing with it, proposed PEB phase in the chapter is simple, quick and effective.

In the IGA, all the LCs are generated by genetic operation from feasible parents chromosomes. There is not any LC created from the two infeasible parents. So the lethal degree of LCs is not stacked and accumulated for multi-times. These types of LCs are lethal due to minority genes rather than majority genes. Therefore the vaccine schema $s_1s_2\ldots s_n$ been trained from many LCs has the feature of excellent chromosomes instead of poor chromosomes actually. That is also the reason why the curves of Fig.3.4(1) and (2) close to 100 in the later generations. Vaccination phase of immune operation based on $s_1s_2\ldots s_n$ not only change the some genes from one to zero, but also change other genes from zero to one. It uses the both of the greater value (crest) feature and small value (trough) feature of $s_1s_2\ldots s_n$ for the purpose that the LCs are revived. Beside PEB keeping the *similarity ratio* from excellent chromosome, vaccination phase in immune operation guides the LCs by the other way to move to the stage of optimal chromosomes.

3.4.2 Immune Operation in IGA

In general, for combinatorial optimization problems, the constraints of the problem bound the solution space into two parts, namely, the feasible space, where all of the constraints are satisfied, and the infeasible space, where at least one of the constraints is not satisfied. Moreover, the optimal solution of the problem generally exists near the boundary on the feasible space side. The standard GA activity at only feasible space side, IGA with immune operation expands the activities range to include also the infeasible space side so that GA obtains the more chance closing to the optimal from infeasible solution space.

In IGA the issue to be solved is how to deal with the LCs which impacts the ability of GA for not only MDKP, but also other combinatorial optimization problems. Immune operation provide GA for LCs a manner that issue been solved depends on the characteristics of issue itself. In the other word, immune operation handles and revives the LCs depend on the characteristics of LCs themselves. This manner reference the characteristics of immune system that re-infection is fought back with the memory of primary infections. Of course immune operation in IGA to handle the LCs with not only the first LC appeared in population but also all heretofore LCs. Since the problem of LCs is also involved in other combinatorial optimization problems not only MDKP, immune operation could be also introduced into GA for other combinatorial optimization problems, even for other swarm intelligence algorithm to deal with the infeasible chromosomes.

The heuristic method to handle LCs such as repair operation of P.C. Chu [7] and penalty function such as one proposed by Farhad Djannaty [9] are excellent and effective for GA to MDKP. But the better a work model could be extracted from them for also other combinatorial

optimization problems beside the MDKP. The immune operation proposed in this chapter is easy to be referenced for GA to other combinatorial optimization problems as long as reasonably using characteristics of infeasible individuals.

3.4.3 Conclusion and Future Research

This study discovered an important feature of lethal chromosomes, and proposed IGA to use this feature of lethal chromosomes based on an immune operation which work under an immune though. Appling IGA to MDKP, a large number of testing results indicate that using lethal chromosomes based on immune operation could obviously improve search performance of GA. At same time there are also some works should to be done to improve the performances of IGA deeply.

In the future, in addition to further improve the double islands model, the method of immune operations should also be researched further. If the proposed immune operations are improved to decrease the time complexity and allow rapider operation, the GA will be applicable to a wider range of field.

Chapter 4

A Genetic Algorithm for the Multi-Knapsack Problems

Many unsatisfied solutions also being produced in applying genetic algorithms to solve the multi-knapsack problem due to genetic operations. The unsatisfied solutions are regarded as lethal chromosomes in genetic algorithms. Large numbers of lethal chromosomes might lead to that implementing and searching performance of genetic algorithms comes to degrade. The usual means dealing with the lethal chromosomes is to eliminate it from population; however, evolved lethal chromosomes contain some fruits of evolution, abandoning lethal chromosomes is as same as abandoning available information, and leads to waste of evolving resources. We propose a new method different from last chapter to revive and utilize the lethal chromosomes based on immune theory, and apply it with a double islands algorithm model. To multi-knapsack problem, simulating experiment shows that proposed method could effectively improve the performance of genetic algorithms.

4.1 The Problem of Lethal Chromosomes

GA is widely applied to the large-scale combinatorial optimization problems for its global search performance and robust performance. With representing the solution of problem as chromosome, through genetic operations such as crossover and mutation, GA achieves searching in space of solutions, then it revert the best chromosome to the solution of the problem.

Actually, many optimization problems have constraints. As for constrained optimization problems GA searches the feasible solutions satisfying the constraint conditions with objective function in whole genetic space. The chromosomes unsatisfying the constraint conditions are called lethal chromosomes.

In a population pool of GA, due to crossover and mutation operations, lethal chromosomes are produced at a high rate, which occurs very much especially to the combinatorial optimization problems having a rigorous constraint. The more of lethal chromosomes appear in the population, the worse the searching performance of GA is, at worst case the algorithm would stop on some occasions.

Researches focusing on problems of lethal chromosomes are still rare. In literature [5], the author had investigated the effects of lethal chromosome to performance of GA, but hadn't supplied a method to deal with it.

If the lethal chromosome appears complying some rules, people could avoid from creating the lethal chromosomes via some means, but for mostly of status, it's difficult to design an algorithm to avoid from producing lethal chromosomes in population. The usual means to deal with the lethal chromosomes is eliminating it from its population and then to supply the new chromosomes created randomly.

In the other way, having evolutions of several generations, lethal chromosomes containing some excellent genes; it is wasting evolving resource to abandon the lethal chromosomes. If GA uses the lethal chromosomes instead of eliminating them, the search performance of the algorithm would be improved.

Literature [6] proposed an algorithm model called double islands model to revive lethal chromosome by crossing and mutation operations randomly. Due to its randomness without using characteristic information, the efficiency of the double islands model algorithm needs to be improved.

In this chapter we propose a method to revive and utilize the lethal chromosomes in GA based on artificial immune theory combining the evolutional information of the chromosome and characteristic

Table 4.1 Notations used in this chapter

Notation	Meaning
m, n	The number of knapsacks and objects
I, J	$I = \{ 1, 2, \dots, m \}$, $J = \{ 1, 2, \dots, n \}$
v_j	The value associated with object j
w_j	The weight of the object j
c_i	The capacity of knapsack i
x_{ij}	The decision variable with object j being selected into knapsack i or not

information of the problem. Appling the algorithm to the constrained combinatorial optimization problem of multi-dimensional knapsack problem (MKP), the numerical experiment shows the ability of the proposed algorithm.

The notations used in this chapter are listed in the table 4.1 for readers to refer expediently.

4.2 An Immune Genetic Algorithm for the MKP

4.2.1 Algorithm Model

Here we explain the proposed GA model which uses the lethal chromosomes based on Immune Theory (IGA). IGA is a modified and improved Simple GA (SGA).

In the GA, a candidate of solution of MKP is represented as chromosome, which is a string of symbols $g_1 g_2 \ldots g_n$, where

$g_i = j$ *when* $x_{ij}=1$

$= 0$ *otherwise (when all* $x_{ik} = 0, k \in I$*)*

g_i is a gene of the chromosome of locus i. This coding system is one of a multiple coding system, which is better than binary coding system according to our experience in literature [6]. The total sum obtained from Eq.(3.1) for each individual is the fitness of the chromosome. If a chromosome violates the constraint of Eq.(3.2) for some knapsacks, the chromosome is lethal.

In SGA; for crossing operation pairs of chromosomes are selected by roulette method and apply point crossing operation to each pair of chromosomes at randomly chosen locus point. And for mutation operation, apply point mutation to a chromosome replacing the value of gene by random number of $0 \sim m$ on randomly chosen locus point at some mutation rate. When lethal chromosomes appear on applying these operations, SGA discards these lethal chromosomes and retries the operations to produce enough number of chromosomes for next generation pool. The selection method to generate next generation pool is performed by roulette method when obtained chromosomes are enough to select.

The proposed GA, using lethal chromosome based on immune operation (IGA), has two types of population pool, the one is called living island containing chromosomes which satisfy constraints, the other is called lethal island containing lethal chromosomes. In the living island chromosomes are evolved by operations of GA, and in the lethal island lethal chromosomes are revived by immune operations.

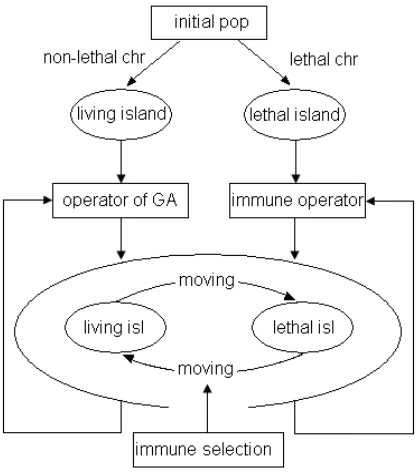

Fig. 4.1 Flow chart of double islands model

In IGA model, after initializing population, the population is divided into two islands according to whether the chromosome is lethal or non-lethal. In the living island lethal chromosomes created by genetic operations are moved to the lethal island. In the lethal island, chromosomes revived by immune operations are selected and moved to the living island. Flow chart of this double islands model is shown by Fig.4.1.

4.2.2 Immune Operation in Genetic Algorithm

Immune operations to revive a lethal chromosome consist of three operations; extracting vaccine, vaccination and immune selection.

(1) *Extracting Vaccine Operation*

This operation is taking out vaccine from the excellent chromosome. Perform the following steps for all lethal chromosomes in the lethal island;

1). Choose a lethal chromosome from the lethal island.

Set the lethal chromosome $l_1 l_2 \ldots l_n$

2). Choose excellent chromosome from the living island by roulette method.

Set the excellent chromosome $e_1 e_2 \ldots e_n$

3). Find all overloaded knapsack numbers.

Find k's for which $\sum_{j=1}^{n} w_j l_{jk} > c_k$

4). Construct a vaccine schema for the lethal chromosome. The vaccine schema is a string of 0, 1.

Set the vaccine schema $s_1 s_2 \ldots s_n$

$s_j = 1$ *when* $l_j = k$ *or* $e_j = k$ *for all k's found above*

$= 0$ *otherwise*

(2) *Vaccination Operation*

This operation is to construct a vaccinated chromosome from the lethal chromosome by changing the genes l_j according to the vaccine for the lethal chromosome.

Replace l_j by e_j where s_j =1

(3) *Immune Selection*

This operation is to select vaccinated chromosomes whose fitness are enough better for being used by GA and move them to the living island.

Revive the vaccinated chromosome when its fitness is better than the average of the living island.

For example, chose a lethal chromosome and an excellent chromosome are as followings:

lethal: 2’ 0 1 3 2’ 0 3 1

excellent: 3 2 0 1 0 2 1 3

In lethal chromosome, “ ’ ” notes object $1^{\#}$ and $5^{\#}$ make the knapsack $2^{\#}$ overloading, so vaccine are e_1, e_5 of excellent chromosome and knapsack $2^{\#}$.

Vaccination is two steps, instead of l_1 and l_5 of lethal chromosome with $e_1$and e_5 of excellent chromosome, we would get:

reviving: 3’ 0 1 3 0’ 0 3 1

For knapsack $2^{\#}$, according that e_2 and e_6 is “2” in excellent chromosome, set l_2 and l_6 of lethal chromosome to *2*, so we could get:

revived: 3’ 2 1 3 0’ 2 3 1

4.2.3 Steps of the Genetic Algorithm with Immune Operation

In the process of reviving and reusing of lethal chromosomes based on double islands model, chromosomes lying in living island evolve by genetic operations, the process is as same as simple GA. Chromosomes lying in lethal island are revived by the immune operators, its process is as follows:

(1). *Initialize population*

Evaluate fitness of chromosomes in the population. Move the non-lethal chromosomes into the living island and the lethal chromosomes into the lethal island.

(2). *Evolve population*

(In the living island):

All of the chromosomes in the living island evolve into the next generation by genetic operations (selection, crossover, mutation), and the new lethal chromosomes move to lethal island.

(In the lethal island):

Each chromosome lying in the lethal island is vaccinated and revives when the fitness after vaccination becomes higher than average of the living island, and moves to the living island.

(3). *Repeat step (2) until the termination condition of the GA is satisfied.*

4.3 Simulation Experiments

Large scale combinatorial optimization problems are real necessary for GA to solve. So we tested IGA on a large scale MKP that number of objects is 1000 and number of knapsacks is 50. The profit associated with objects creates randomly between 1000 and 10000, consumption of knapsacks for object is between 100 and 1000. For 50 knapsacks, capability also create randomly between 1000 and 10000.

Set population sizes is 100, To compare with SGA, we run program of IGA and SGA respectively under the same status with PC condition of CPU 2.0GHz.

In order to avoid the effects from occasional factors of experiments, we have run the proposed algorithm IGA and SGA 10 times under different random number seed. But we ensure that IGA and SGA to evolve starts from the same initial population and under the same parameter of GA in same time. Therefore the means of experiment we adopt ensure a credible, steady result for our experiment.

4.3.1 Experimental Results Comparing with GA

Table 4.2 shows the number of chromosomes revived and effective rate changing along generation for one try of IGA. The effective revived chromosomes are stated as that chromosomes revived have a fitness which is better than the average of the living island. It is also that only effective revived chromosomes would be selected moving to the living island for utilizing.

Table 4.2 Effective revived rate

generation	100	200	300	400	500	600
reviving	25	54	91	91	93	92
effective	11	30	45	91	93	92
effective rate	0.44	0.56	0.49	1.0	1.0	1.0

From Table 4.2, firstly, it is clear that the more closely population evolving toward the optimal solution, the higher rate the genetic operations creating the lethal chromosomes, moreover the more lethal chromosomes been revived. It implies necessity

of reviving lethal chromosomes. Secondly, not only more lethal chromosomes were revived effectively but also it occurs at a higher effective rate, and all of these changing under process of population being close to the optimal solution.

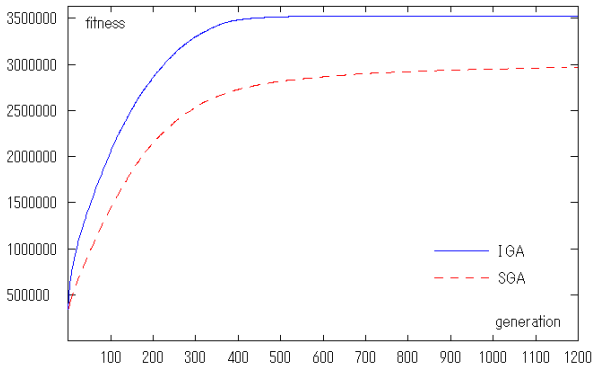

Fig. 4.2 Evolution curve of best solution

For average of 10 times, comparing with SGA, the evolving curve of proposed IGA showed by Fig.4.2.

From this figure, the converged optimal solution found by IGA is better than that of SGA and shows the very rapid evolution by IGA. IGA could reach to its optimal solution before 600th generation, but SGA still has not converged to its optimal solution at 1200th generation. Duration of SGA is 43.81 seconds, but when 600th generation IGA's duration is only 26.43.

4.3.2 The Ability to Obtain the Exact Solution

In order to test the ability of reaching to the exact optimal solution, we experimented on another set of MKP that the number of objects is 15, knapsacks is 3, and it is easy to get the exact optimal solution by the classical algorithm.

Table 4.3 Reaching times with running the algorithm 100 times

population	5	10	20	40	60	80
SGA	1	12	23	40	53	61
IGA	14	45	65	81	92	99

Table 4.3 shows the times of reaching to the exact optimal solution with running the algorithm 100 times and terminating at 100th generation. It is clear that IGA exceeds SGA so much.

4.4 Discussion

During the last two decades, solving combinatorial optimization problems with GA has attracted the attention of many researchers. However, researches on lethal chromosomes for improving performance of GA are still rare. Literature [5] [6] pay attention to the problems and published more than decades ago. This chapter proposes a method to revive and utilize lethal chromosomes.

4.4.1 Algorithm Model and Immune Operation

The algorithm model chapter adopt is double islands. It is mainly different from SGA that the double islands model partitions the population into two groups stated as the living island and the lethal island. It leads to the algorithm to increase the spatial resources to save the lethal island, but it is not necessary to save the lethal chromosomes in SGA. That also increases the spatial complexity of algorithm comparing with SGA. Maybe we can find out a better way instead of double islands to solve the problem well without increasing the spatial complexity. We are doing researches in this area.

It is very clear that eliminating the lethal chromosomes is easier than to revolve it. But abandoning the lethal chromosomes after evolution of several generations is equal to discard the resources of evolution. In order to revolve the lethal chromosomes we vaccinate them with immune operations. It spends more time for every lethal chromosome to be vaccinated. It also increases the time complexity for algorithm from SGA.

Although IGA have more complexity than SGA, due to immune operations based on double islands model and utilizing the characteristic information of problem, the performance of GA is improved obviously. We could learn from the results of experiments that the IGA improves the optimal solution and the speed of evolution from SGA.

4.4.2 Conclusion and Future Work

To constrained combinatorial optimization problems, defining chromosome with concept of Fuzzy logic instead of regarding it as only lethal or non-lethal, is another way to solve the problem. All of chromosomes based on Fuzzy logic will have the opportunity to take part in the competition of evaluation according its Fuzzy fitness. IGA with Fuzzy logic can avoid some troubles of increasing complexity to algorithm, and may lead the problem to a new case. We are now preceding the IGA by this way.

This chapter presents a method for reviving lethal

chromosomes based on immune operations and double islands. Appling it to MKP indicates that this method could obviously improve search performance of GA. In the future, besides necessity of double islands model to be improved for farther, it is important that method of revolving based on immune operator needs to be researched deeply.

If proposed GA would be improved in decreasing the time complexity and spatial complexity, GA will be applied in more fields such as real-time control and embedded system. That is there are more works to be done for applying GA to more fields.

Chapter 5

A Genetic Algorithm for the Stock Layout Problems

Metal curtain wall consists of many small pieces of plate in large-scale building decoration projects. The small pieces of plate always resemble rectangular shape, so stock layout of rectangles, which also be called two-dimensional cutting problem, is often faced in the actual projects. In addition guillotine cutting is a technological requirement as plate work by machine processing. Therefore stock layout of rectangles is a constrained optimization problem. This chapter proposing a matching segmentation algorithm and a genetic algorithm combines local searching and global searching for the solution of stock layout with constraint of guillotine cutting. Appling the proposed algorithm to practical projects shows its validity.

5.1 Stock Layout Problem in Practice

We were commissioned by a factory to research layout stock for the purpose of solving real production problems. The problem is defined according to the requirements of the production process, and the objective function (1) is designed according to that definition. Although some prior studies use the same definition of the problem as the present chapter, we could find few studies using the same objective function.

A layout pattern often includes more than one stock whether the size of stocks is single or multiple. If the area of all items demanded is defined as U_1, and the area of all stocks included in the layout pattern is defined as U_2, then the objective function is typically defined as the ratio of areas U_1/U_2.

However, in most cases, the last piece of stock in a layout pattern is not fully used because all the items have been cut. In other word, the last piece of stock has a remainder. Therefore, we designate the remainder area as R and subtract it from U_2, as shown in formula (2.6). The purpose is not to inflate the percent of utilization, but to improve the optimization.

These are not simply interesting NP-complete mathematical problems but are also relevant to many other technological requirements arising in actual production operations. The most common of these requirements can be summarized as the following:

(1) *Guillotine cutting.*

In practical production processes, plate work for cutting stocks is often done by machinery in such a way that cutting the sheet metal stocks often requires a layout pattern consistent with guillotine cutting. This requirement arises because the cutting process is almost impossible to stop at a precise point internal to the stock. This makes it difficult to avoid short-cutting or over-cutting, where the former makes it difficult to separate the parts, the latter leads to damaging other sub-pieces.

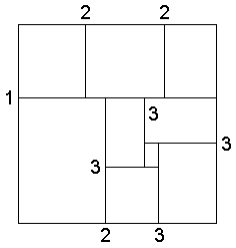

Fig. 5.1 An example of layout pattern with guillotine cutting lines and non-guillotine cutting lines.

Guillotine cutting is defined as cutting such that every cutting operation divides the stock into two pieces; in particular, the cutting line runs from one edge of a rectangle to the opposite edge, parallel to the other two edges. For example in Fig.5.1 the cutting

Table 5.1 Notation used in this chapter

Notation	Meaning
m, n	The number of stocks and the items, respectively
L, W, T	Length, width and texture direction for a type of stock
l, w, t	Length, width and texture direction for a type of item
m_c	The number of stocks being chosen in a layout pattern
R	The residual area of the last piece of stock used in the layout pattern
$A_1A_2\ldots A_m$	A sequence of all available stocks
$B_1B_2\ldots B_n$	A sequence of all items demanded
$x_1x_2\ldots x_n$	A 4-value string to indicate the way placing and cutting for stocks.
$y_1y_2\ldots y_n$	A 4-value string to indicate the way placing and cutting for items.
Decoder	Obtain a layout pattern solution based on individual

line “1” is a guillotine cutting line and all the cutting lines “2” and “3” are not. But after the stock is cut away with line “1”, in the two pieces of sub-stocks the lines “2” will become to the guillotine cutting lines. But the lines “3” cannot become to the guillotine cutting lines in anyway. The guillotine cutting requirement is an additional constraint on the general stock layout problems.

(2) *Texture matching.*

Matching texture direction for stocks and items is often required. For example, the glass cutting requires that the glass items should be installed in windows with a standing portrait pattern. And wood cutting also sometimes requires a fixed texture direction for items depending on the texture of the stocks.

(3) *Width of kerfs.*

The cutting process results in certain widths of kerfs; which must be taken into account in the layout pattern. If accommodating width of kerfs is done only via enlarging the size of items in advance, some items will have final sizes that do not match the required sizes. This case will be explained in more detail below.

(4) *Adapting to strip-type stocks.*

Strip-type stocks are plane materials which have fixed width and effectively unlimited length, for example, soft sheet material rolled into cylinders, such as cloth. The algorithm should also be adaptable to strip-type stock, not just to stocks having given length.

(5) *Choosing stocks.*

In general, the layout pattern depends on the material inventory. For example, there may be multiple sizes of stock that can be used rather than a single size. The algorithm should be able to handle choosing the optimal stock size from available sizes to minimize waste.

Many researchers proposed the algorithm to stock layout problem with various definition of problem described as section 2.3.1. This chapter establishes a problem model including requirements (1) to (5) and proposes a coding method and a fish swarm optimization (FSO) algorithm. In practical application, we can remove any one or more of the requirements as a special case of the problem model if this requirement is not necessary. For example, we can regard the stocks as having no texture direction to consider the model without requirement (2); set the width of kerfs as 0 to exclude requirement (3); and let the stocks be of only one size to exclude requirement (5). However, a good optimization algorithm should be able to cope with all the requirements. This chapter proposes an approach based on FSO algorithm to solve the guillotine cutting problems with these requirements.

The notations used in this chapter are listed in the table 5.1 for readers to refer expediently.

5.2 A Genetic Algorithm for the Stock Layout Problems

The chapter proposes a representation and decoder to achieve a solution of layout. The representation works depending on a sequence of stocks and items given, besides a set of orientations associating with each item. As these sequence and orientations would decide the result of layout by decoder, we designed a data structure containing information of sequence and orientations to be optimized by GA. In GA this data structure is also named as chromosome, and we will state its detail of GA in the later section.

5.2.1 Representation

In any case, when the first item is placed in empty stock, we bet that place the item on bottom-left corner. The first item must be placed along a guillotine line, so we locate first guillotine cutting along the side of this item. How to place the item and how to locate guillotine cutting line, there are four cases. As Fig.5.2, (1) shows the horizontal placing & vertical cutting, (2) the vertical placing & vertical cutting, (3) the horizontal placing & horizontal cutting, and (4) the vertical placing & horizontal cutting.

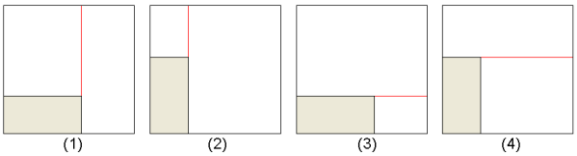

Fig. 5.2 Four cases how to place and how to cut.

Obviously, orientations of placing and cutting immediately influence the result of layout solution. Therefore, not only the sequence of stocks and items but also orientations of placing and cutting, we consider it as information of chromosome which is translated to layout solution by decoder.

Input string to be translated into layout by decoder is designed as a two-line chromosome:

$$\begin{pmatrix} A_1 & \cdots & A_m & , & B_1 & B_2 & \cdots & B_n \\ x_1 & \cdots & x_m & , & y_1 & y_2 & \cdots & y_n \end{pmatrix}, \; x_i, y_j \in \{0,1,2,3\} \quad (5.1)$$

where $A_1 \ldots A_m$ provides a sequence of stocks, $B_1 \ldots B_n$ provides a sequence of items. In second line, $x_1 \ldots x_m$ denotes whether rotated 90° for each stock before being filled by items. If x_i is '1' or '3', it indicates rotating A_i by 90°, otherwise '2' and '4' indicate without rotation. $y_1 \ldots y_n$ denotes orientations of being placed and cutting line for each item. If y_j is '0', it denotes placing B_j without rotation & vertical cutting, '1' indicates rotating by 90° & vertical cutting, '2' indicates placing without rotation & horizontal cutting, '3' indicates rotating by 90° & horizontal cutting.

5.2.2 Decoder Function

To a stock, all of items could be classified two types. *Matching item*: items whose one of edge as same as one of edge of the stock. *Segmentation item*: items which isn't matching item to the stock.

For achieving guillotine cutting, the decoder searches all of matching items according an order given by chromosome firstly, and place items onto stock one by one, pick a segmentation item based on sequence given by chromosome and place it on bottom-left of stock, then locate a guillotine cutting line beside of item placed, so that the stock was separated into two sub-stocks, and the problem is reduced to layout of two pieces of sub-stocks. The steps of decoder function outlined below.

(1). *Read sizes of stock A_i as l and w, add a width of kerfs to l and w (adding a width of kerfs to both length and width for all items beforehand).*
(2). *If all of items were used, return.*
(3). *Search all matching items of this stock and place them onto stock aside, l(or w) be cut down.*
(4). *Pick an segmentation item B_j in turn beads on chromosome, place (and rotate 90°)[1] it on bottom-left of stock, separate the stock into two sub-stocks along vertical (or horizontal)[1] edge of item placed.*
([1]decided by correlative y_j of chromosome)
(5). *Read length and width of first sub-stock as l and w, push sizes of second sub-stock into stack, return step 2.*
(6). *Sizes of second sub-stock pop stack as l and w, return step 2.*

If add a width of kerfs only to sizes of items without to stock, after items separating from its stock, some stocks, which be placed along edges of stock, are bigger than real required size at length or width, step.1) avoids this problem.

5.2.3 Genetic Operation

The chromosomes are optimized by GA. We designed the GA operations of crossover, mutation, and selection regarding the coding method of chromosome.

(1) *Chromosome.*

GA describes the feasible solution as a string called chromosome. It also provides references to the decoder function for achieving the solution. The chromosome should contain all factors affecting result of layout. In literature[30], chromosome could not reflect orientations of placing items, it decides the orientations of placing only with matching algorithm, effects the global capability of GA. To a solution of layout, chromosome should decide how to select stocks, orientations of placing items and orientations of cutting line. Therefore we designed a two-line chromosome as Equation (4.2).

(2) *Crossover Operation.*

Pairs of chromosomes are selected by roulette method. We apply point crossing operation to each pair of chromosomes at randomly chosen locus, and apply point mutation to a chromosome. The chromosome contains two types of information, sequences and orientations. The crossover must achieve exchanging both sequences and orientations. We designed two means of crossover.

Sequence crossover: selecting a point within length of chromosome, exchanging part of sequences of stocks or items according crossover point. To stocks and items, with changing part of sequence, other part of sequence should be refreshed for keeping original sequence. In this means of crossover, for each stock and item, orientations noted as x_i and y_j is unchanged.

Orientations crossover: exchanging information of orientations between pairs of chromosomes each other. That is changing corresponding x_i and y_j for each stock and item according another chromosome.

(3) *Mutation Operation*

Mutation is selecting two points within $A_1 \cdots A_m$ (or $B_1 \cdots B_n$) randomly, then exchanging them, at same time exchanging the correlative x_i (or y_j). Another means of mutation is randomly selecting a point of second line of chromosome, then changing it within {0, 1, 2, 3} randomly.

(4) *Selection Operation*

For Next Generation In evolution process of GA, after crossover and mutation operation to every pairs of chromosomes, we adopt the roulette-wheel selection method to form next generation chromosome pool.

5.3 Examples of layout Pattern

For simulation experiments, setting the population of GA as 100 and the max-generation as 500, we run our algorithm on a windows-XP System with 2.00GHz Clock of CPU and 2.0GB RAM.

5.3.1 Examples for Set Layout

In order to compare our algorithm with other algorithms, we have experimented on mass test data provided by literatures with guillotine cutting. As means of obtaining guillotine cutting problems, proposed algorithm achieves a better solution at utilization rate of stocks and rationality of layout.

Here we give the computational example of layout pattern to project data drown from literature [53] and literature [33] as Fig.5.3 - 5.6. For Fig.5.3 and Fig.5.4, sizes of stock are 600mm (length) and 400mm (width), the amount is enough to use; items for layout as Table 5.2. For Fig.5.5 and Fig.5.6, sizes of stock are 4000 mm(length) and 2900mm(width), the amount is enough to use too; items for layout as Table 5.3.

Comparing Fig.5.3 with Fig.5.4, and comparing Fig.5.5 with Fig.5.6, the both algorithms used two pieces of stocks. The result by the proposed method is more reasonable as it produces a larger piece of residue which may be effectively used again for next time in second piece of stock. So the algorithm proposed achieves a better solution.

5.3.2 Examples for Strip-type Stock

The proposed algorithm is also suitable for strip-type stock. We have simulated with project data of literature [54] that width is 600 and length is boundless, items for layout as Table 5.4. The result is showed by Fig.5.8. Literature [54] improved a pack algorithm to a genetic simulated annealing algorithm, so that reduced the length of stock used from 1260 to 1245 showed as Fig.5.7, the present algorithm makes the result to 1220. It is clear that the algorithm exceeded.

We have also performed simulation experiments from famous OR-Library [36] which is a collection

Table 5.2 The items of taking part in layout from literature[53].

Serial number	Length(mm)	Width(mm)	Items
1	130	70	3
2	140	80	3
3	140	100	1
4	220	210	1
5	240	90	3
6	300	70	2
7	120	80	3
8	350	90	3
9	310	130	1
10	130	110	3

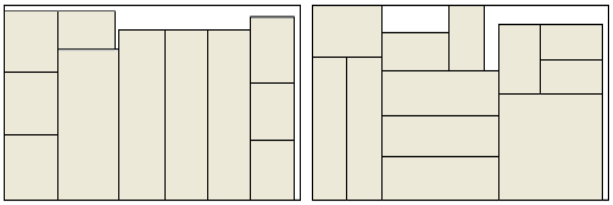

Fig. 5.3 Layout pattern from Ref. [53]

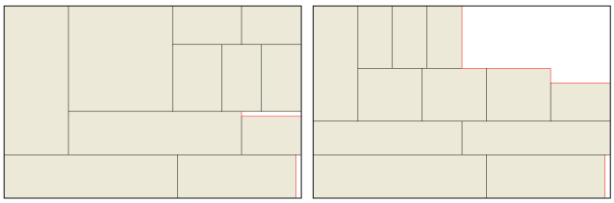

Fig. 5.4 Layout pattern by GA for instance of Ref. [53]

Table 5.3 The items of taking part in layout from literature[33].

Serial number	Length(mm)	Width(mm)	Items
1	450	60	103
2	500	100	70
3	540	70	90
4	400	100	120
5	200	100	150
6	360	150	40
7	450	120	45

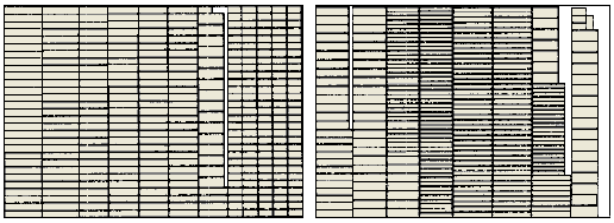

Fig. 5.5 Layout pattern from Ref. [33]

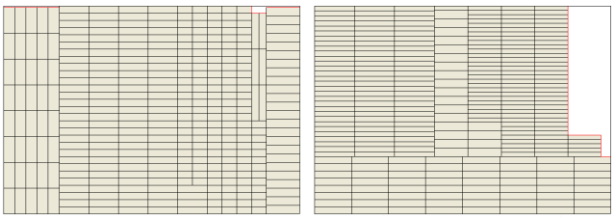

Fig. 5.6 Layout pattern by GA for instance of Ref. [33]

of test data sets for a variety of OR problems. It is "Bin packing - two dimensional" of OR-Library for this chapter adopted. This collection provided 15 instances including three categories for this problem. For first categories containing 5 instances, our algorithm obtains 100 percent of utilization rate. For second categories we obtain an average of 98.033 percent, third categories is 98.210 percent. Although

Table 5.4 Sizes and number of items from Ref. [54]

Serial Number	Length (mm)	Width (mm)	Items
5	40	20	40
8	15	6	78
12	50	30	19
1	50	25	59
6	50	40	64
9	18	6	35
15	45	15	26
2	35	15	49
3	60	40	66
4	80	40	80

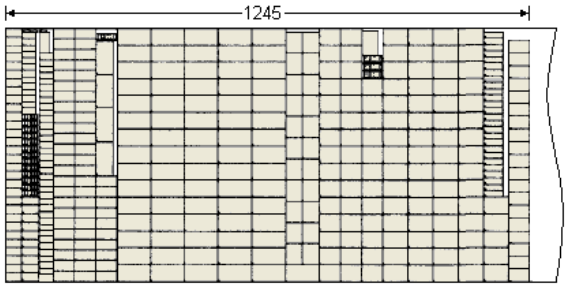

Fig. 5.7 Layout pattern from Ref. [54]

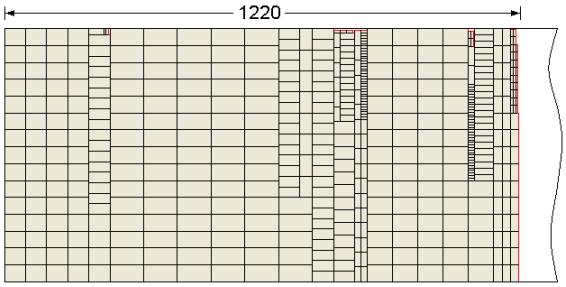

Fig. 5.8 Layout pattern by GA for instance of Ref. [54]

no optimal solution is known, it is obvious that our algorithm achieves a least waste of stocks.

5.4 Discussion

For the stock layout problem many scholars had done important research works. Many excellent literatures solved these problems well. Comparing with these literatures this chapter has many differences.

5.4.1 Constraints of Guillotine Cutting

Requirement of guillotine cutting is a constraint to optimize layout actually, it will greatly reduce the evaluation of solution which associates with utilization rate of stocks. How to meet guillotine cutting as well as optimize the solution of layout is one of the viewpoints of this chapter.

Actually it is difficult that all cut lines are guillotine line in a stock. If there is only one guillotine line in the stock, we can cut the stock into two sub-stocks with this guillotine line. As well as if there is also only one guillotine line in two sub-stocks respectively, we can cut them again. According the algorithm, the items are placed onto stock as matching item or segmentation item. When a matching item is placed one guillotine line is settled, also when a segmentation item is placed a guillotine line appears aside. A stock having on matching item will be cut by the guillotine line which is located by a segmentation item, and it is an only guillotine line to this stock. In this case, the algorithm makes a least guillotine line in a stock and its sub-stock. This is one of the important points for this chapter, and it greatly decreases the constraints of guillotine cutting so that we achieve a fine solution at utilization rate of stocks and rationality of layout.

5.4.2 Applicability of Algorithm

The algorithm is designed applicable to a variety of specification and multi-quantity sheet for stocks. Of course, single stock is also applicable as a special case. Another specific stock is the case that with fixed width and variable length, we called it strip-stock. By controlling the chromosome we could achieve the layout for strip-stock, which is another special case to multi-specification & multi-quantity stock, and belongs to case of single sheet. So our algorithm is suitable for all cases of thick sheet which needs being cut by guillotine line.

5.4.3 Global Feature of the Solution

According the algorithm proposed, each chromosome reflects a unique solution of layout by decoder function. Contrarily, to any case of layout solution meeting guillotine cutting, we are able to find out the chromosomes reflecting it in genetic space. Because the coding method is considered for all of factors which affect result of layout, any case of solution is expressed by chromosome space; the

proposed algorithm can find out very fine solution at a great possibility. So combination of decoder function and GA ensure that this algorithm work well in the genetic space.

5.4.4 Cutting efficiency of the Layout Pattern

In decoder function, items are placed onto stock as matching items or segmentation items. At first case, when the matching items are placed at an end of stock, it is very possible that the next same sizes items are placed along the last one at residuary space of stock for its matching size. In second case, after placing a segmentation items at bottom-left of stock, stock will be cut into two pieces, so that one of the sub-stocks has a same edge with the segmentation items. So it is also very possible that the next same size item would be placed in sub-stocks as a matching item. In this way, the same size items would be placed together at a great possibility. It is very helpful to process of cutting, as well as greatly improved efficiency of cutting process without influencing the evaluation of solution.

Chapter 6

An Improved Fish Swarm Optimization for the Stock Layout Problems

The stock layout problem is an NP-Complete problem applicable not only to economizing materials, but also to meeting many technological requirements of production processes, including guillotine cutting, texture matching, width of kerfs, adaptation to strip-type stocks, and selection of stocks from among multiple sizes. This chapter considers all of these requirements as constraints of the problem and proposes a description of the problem and a fish swarm optimization method to solve the problem under these constraints. The proposed algorithm is applied to a large number of test data and practical projects, the results are compared with those of other algorithms to demonstrate its superiority.

6.1 Behavior Operations in Fish Swarm Optimization

The stock layout problem is described as previous parts chapter 2 and chapter 5. The problem for this chapter to solve is also what was described in last chapter for GA to solve, including the guillotine cutting and other process requirements which were summarized as requirements from (1) to (5). To same definition of problem with last chapter, this chapter will test the FSO and GA on large number of same instances and compare with the results each other.

This chapter focuses on the application of Fish Swarm Optimization (FSO) to stock layout. As described in chapter 2, the problem of convergence and computational complexity about FSO are the defeats in the current. Especially the computational complexity seriously affects the evolutionary process. As mentioned in chapter 2, FSO has the feature of fast convergence, but that mainly depends on step of process. If the computational complexity could be improved, the time complexity will be saved. In the context that speed of convergence is guaranteed, step of process could be set smaller, thus the problem of convergence will be improved.

In the FSO solving the stock layout, the main computational complexity should be ascribed to three behavior operations foraging, clustering and following. The most one to cost the computational complexity is clustering operation which has to calculate the center position of other partners. During the clustering operation, not only every partner present individual can sense in near environment should be counted, but also present individual coding should be checked as a correct coding data. On the other hand, it is also mainly cost the computational complexity to judge the crowding degree for the near environment. Focusing on these problems, this chapter optimizes the behavior operation and operation selection rule, proposes a improved application for FSO to solve the stock layout.

6.2 An Improved Fish Swarm Optimization for the Stock Layout Problems

Notation and functions to be used in the text are listed

Table 6.1 Notation and functions used in this chapter

Notation	Meaning
m, n	The number of stocks and the items, respectively
L, W, T	Length, width and texture direction for a type of stock
l, w, t	Length, width and texture direction for a type of item
m_c	The number of stocks being chosen in a layout pattern
R	The residual area of the last piece of stock used in the layout pattern
$A_1A_2\ldots A_m$	A sequence of all available stocks
$B_1B_2\ldots B_n$	A sequence of all items demanded
$x_1x_2\ldots x_n$	A 4-value string to indicate the way of placing and cutting for each item.
F	The individual coding data, consisting of $A_1A_2\ldots A_m$, $B_1B_2\ldots B_n$, and $x_1x_2\ldots x_n$, as show in formula (5.1)
F_i	The i-th individual coding data
k	The number of individuals
$C(F_1,\ldots,F_k)$	The function which returns the individual coding data that is the center of $F_1, \ldots, F_k$
$D(F_1,F_2)$	The function which returns the distance between F_1 and F_2
r, rm	The perception range of an individual, and the number of individuals within range r
Decoder(F)	Obtain a layout pattern solution based on individual F

in Table 6.1. They will be explained at first use in the text. However, the reader may find it more convenient to look up definitions in the Table 6.1.

6.2.1 Algorithm Model

The idea of FSO is to simulate the process of fish swarm foraging [50]. In oceans or lakes, fish swarm typically can find nutrient-rich areas quickly. Observing the behavioral characteristics of fish swarms, fish activities feature clustering behavior, following behavior, and foraging behavior. This chapter simulates these features as algorithm operations to optimize the stock layout problems.

The algorithm model of FSO uses the model in the same way as most swarm intelligence algorithms, such as that of the genetic algorithm (GA). That is, first it analyzes the problem and determines the coding method for optimization of individuals (artificial fishes). Next, it generates and initializes the original population of individuals. And then, according to the behavioral characteristics of the fish in the swarm, performs the optimizing iteration of the population using the algorithm operations. Finally, it decodes the best individual obtained as the layout pattern solution. Since the GA and PSO models are well known to researchers who work with optimization algorithms, the present chapter does not re-introduce them in detail.

6.2.2 Individual Representation

An individual as an optimizing object is coding data which corresponds to a potential solution of the problem in most swarm intelligence optimization algorithms. We analyzed the factors impacting the results of the solution and code them into a data structure called individual F according to formula (5.1). F may be decoded into a layout pattern solution by the decoder function $Decoder(F)$, which will be described in detail later.

$$F:\begin{pmatrix} A_1 & \cdots & A_m, & B_1 & B_2 & \cdots & B_n \\ & & & x_1 & x_2 & \cdots & x_n \end{pmatrix},\ x_i \in \{0,1,2,3\} \quad (6.1)$$

(1) To define choosing stocks from storage, it is sufficient to give a sequence of various types of stocks such that stocks are used in turn until all of the items are cut from the stocks. So F should include a sequence of stocks $A_1A_2\ldots A_m$.

(2) We also need a sequence of items based on which items are assigned to stocks, in order. The combination of these two sequences achieves the selection for stocks to items. Therefore F should also include the sequence of items $B_1B_2\ldots B_n$.

(3) Besides which stock an item should be assigned to, it is also necessary to specify how to set the guillotine lines. Therefore, F includes the orientation of the guillotine lines as a sequence of four-value variables $x_1x_2\ldots x_n$. When an item is assigned to an empty stock, the item is assigned to the bottom-left corner. To make a cut separating the first item via a guillotine line, we must locate the first guillotine cutting next to the first item.

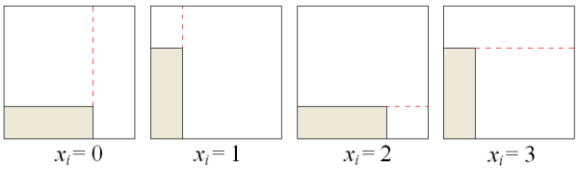

Fig. 6.1 Four ways to orient item and guillotine cutting

There are four possible ways to orient the item and the guillotine cutting line: (i) horizontal item orientation and vertical cutting, (ii) vertical item orientation and vertical cutting, (iii) horizontal item orientation and horizontal cutting, and (iv) vertical item orientation and horizontal cutting. So we denote these using a sequence $x_1x_2\ldots x_n$ of variables x_i (i = 1, 2, …, n) which take values 0, 1, 2, or 3 to differentiate these four cases, as shown in Fig.6.1.

In an individual coding, $A_1\ldots A_m$ provides a sequence of stocks, and $B_1\ldots B_n$ provides a sequence of items. In the second row, $x_1\ldots x_m$ denotes whether items are rotated 90° for each item when it is assigned to the stock. The value "0" for x_i indicates assigning B_i without rotation and making a vertical cut, "1" indicates rotating by 90° and making a vertical cut, "2" indicates assigning without rotation and making a horizontal cut, and "3" indicates rotating by 90° and making a horizontal cut.

6.2.3 Definitions of the Center and Distance

Before getting into the details of the algorithm, we need two definitions. One is the center of a set of individuals. For individuals F_1, F_2,…,F_k, their center position F_c is obtained from the following definition:

$$C(F_1,\cdots,F_k)=\begin{pmatrix} \underset{i=1,\cdots,k}{Most}\ A_1^i & \cdots & \underset{i=1,\cdots,k}{Most}\ A_m^i, & \underset{i=1,\cdots,k}{Most}\ B_1^i & \cdots & \underset{i=1,\cdots,k}{Most}\ B_n^i \\ & & & \underset{i=1,\cdots,k}{Most}\ x_1^i & \cdots & \underset{i=1,\cdots,k}{Most}\ x_n^i \end{pmatrix} \quad (6.2)$$

$$F_i:\begin{pmatrix} A_1^i & A_2^i & \cdots & A_m^i, & B_1^i & B_2^i & \cdots & B_n^i \\ & & & & x_1^i & x_2^i & \cdots & x_n^i \end{pmatrix}, i=(1,\cdots,k)$$

We consider an individual as some data points, such as A_1, B_1, and x_1. For each data point, the center is takes the value which appears for most times in corresponding data point for F_1, F_2,…,F_k. In this way, F_c is determined by all of the data points, but the resulting F_c may not be a valid individual coding, therefore this must be checked, and be ensured that there is no repeat and no missing one in the sequences $A_1A_2\ldots A_m$ and $B_1B_2\ldots B_n$.

The other definition needed is the distance between two individuals. Given F_1 and F_2, the distance is

$$D(F_1,F_2)=\begin{cases} m+n, & \left(A_1^1\neq A_1^2\right) \\ m+n-s_1, & \left(A_i^1=A_i^2,\ i=1,2,\cdots,s_1,\ and\ A_{s_1+1}^1\neq A_{s_1+1}^2\right) \\ & \left(0<s_1<m\right) \\ n, & \left(A_i^1=A_i^2,\ i=1,2,\cdots,m, B_1^1\neq B_1^2\ or\ x_1^1\neq x_1^2\right) \\ n-s_2, & \begin{pmatrix} A_i^1=A_i^2, i=1,2,\cdots,m \\ B_j^1=B_j^2\ and\ x_j^1=x_j^2,\ j=1,2,\cdots,s_2 \\ B_{s_2+1}^1\neq B_{s_2+1}^2\ or\ x_{s_2+1}^1\neq x_{s_2+1}^2 \end{pmatrix} \\ & \left(0<s_2<n\right) \\ 0, & \begin{pmatrix} A_i^1=A_i^2, i=1,2,\cdots,m \\ B_j^1=B_j^2\ and\ x_j^1=x_j^2,\ j=1,2,\cdots,n \end{pmatrix} \end{cases} \tag{6.3}$$

$$F_1:\begin{pmatrix} A_1^1 & A_2^1 & \cdots & A_m^1, & B_1^1 & B_2^1 & \cdots & B_n^1 \\ & & & & x_1^1 & x_2^1 & \cdots & x_n^1 \end{pmatrix}$$

$$F_2:\begin{pmatrix} A_1^2 & A_2^2 & \cdots & A_m^2, & B_1^2 & B_2^2 & \cdots & B_n^2 \\ & & & & x_1^2 & x_2^2 & \cdots & x_n^2 \end{pmatrix}$$

where, individual data is considered as $m + n$ columns. The distance between two individuals is the number of columns counting from first column which differs through to the last column (B_n, x_n). For example, if the first k columns are the same in F_1 and F_2, but the $(k + 1)^{th}$ columns are different, then $D(F_1, F_2)$ counts the columns from the $(k + 1)^{th}$ column to the last column, that is, $(m + n - k)$.

6.2.4 Behavior Operations

In the process of FSO, every individual chooses one of the behavior operations to perform as iteration in population. Setting the current individual as F_i, FSO first sets a parameter r as the perception range for the individual, which is measured from itself according to the definition of distance (4), and then FSO counts the number of other partner individuals in population within distance r whose positions are better than that of the current individual F_i, expressing this number as rm. The goodness of the position is measured with respect to the utilization of stocks (1).

In the each iteration, FSO performs one behavior operation for each individual based on rm it counted. If rm is greater than 1, it performs the clustering operation; if rm is 1, it performs the following operation; and if rm is 0, it performs the foraging operation. These three kinds of behavior operations are described as follows:

(1) Clustering behavior operation

To performance the clustering operation, the rm is greater than 1. The clustering operation first acquires the center position of these rm individuals using (3) to obtain F_c. Then, if F_c is valid as individual data, it replaces F_i with F_c, written $F_i \leftarrow F_c$. The role of clustering operation is to move current individual to other position where the partners have better fitness.

(2) Following behavior operation

In this case, rm is 1, which indicates that there is only one individual at a better position than the present F_i from among partner individuals within its perception range, expressed as F_b.

$$F_i:\begin{pmatrix} A_1^i & A_2^i & \cdots & A_m^i, & B_1^i & B_2^i & \cdots & B_n^i \\ & & & & x_1^i & x_2^i & \cdots & x_n^i \end{pmatrix}$$

$$F_b:\begin{pmatrix} A_1^b & A_2^b & \cdots & A_m^b, & B_1^b & B_2^b & \cdots & B_n^b \\ & & & & x_1^b & x_2^b & \cdots & x_n^b \end{pmatrix}$$

The following operation obtains the distance d between F_i and F_b. Then the algorithm performs an operation reducing $D(F_i, F_b)$ by one ($d \rightarrow d - 1$). Specifically,

$$if\ (0<d\le n)\ then\ B_{n-d+1}^i \leftarrow B_{n-d+1}^b,\ x_{n-d+1}^i \leftarrow x_{n-d+1}^b$$

$$else\ if\ (n<d\le m+n)\ then\ A_{m+n-d+1}^i \leftarrow A_{m+n-d+1}^b$$

After that, there are two repeating same columns and also one missing columns in sequence of stocks or items. It is necessary to set the later repeat one as that missing one to maintain the F_i as a valid individual data. The role of following operation is to move current individual to a better position for one step.

(3) Foraging behavior operation

Given position F_i, try another position F_i' chosen at random such that $D(F_i, F_i') < r$. If F_i' is a better position than F_i, then replace F_i by F_i'. If several tries does not produce a better position, F_i is left unchanged.

The random selection of a new position is achieved by randomly selecting two columns in individual F_i and exchanging the corresponding values, either within $A_1A_2\ldots A_m$ or within $B_1B_2\ldots B_n$ and the corresponding $x_1x_2\ldots x_n$. For example, two columns within $A_1A_2\ldots A_m$ can

be exchanged as in

$$F_i:\begin{pmatrix} A_1 & A_2 \overset{\Leftrightarrow}{} A_3 & \cdots & A_m, & B_1 & B_2 & \cdots & B_n \\ & & & & x_1 & x_2 & \cdots & x_n \end{pmatrix}$$

$$\rightarrow F_i':\begin{pmatrix} A_1 & A_3 & A_2 & \cdots & A_m, & B_1 & B_2 & \cdots & B_n \\ & & & & & x_1 & x_2 & \cdots & x_n \end{pmatrix}$$

or two columns within $B_1B_2\ldots B_n$ and the corresponding $x_1x_2\ldots x_n$ can be exchanged as in

$$F_i:\begin{pmatrix} A_1 & A_2 & \cdots & A_m, & B_1 \overset{\Leftrightarrow}{} B_2 & B_3 & \cdots & B_n \\ & & & & x_1 \overset{\Leftrightarrow}{} x_2 & x_3 & \cdots & x_n \end{pmatrix}$$

$$\rightarrow F_i':\begin{pmatrix} A_1 & A_2 & \cdots & A_m, & B_2 & B_1 & B_3 & \cdots & B_n \\ & & & & x_2 & x_1 & x_3 & \cdots & x_n \end{pmatrix}$$

The role of foraging operation is to achieve mutation and breakthrough in population.

6.2.5 Decode Individual into Layout Pattern

With iteration of the population, we should decode the best individual into a layout pattern. *Decoder*(*F*) selects the items for stocks and arranges locations of items and the orientation of the guillotine cutting line within the stocks according to individual coding *F*.

Given a stock, for the first item, all items are classified into two groups. *(i) Matching items*: items having one edge coinciding with the same edge of the present stock. *(ii) Non-matching items*: all items that are not matching items. The steps of *Decoder*(*F*) are outlined below:

1. *$i \leftarrow 1$*
2. *From stock A_i, obtain the L, W, and T from the sequence provided by F and add the width of the kerfs to L and W. (Note, the width of kerfs are already added to l and w for all items beforehand).*
3. *If there are no any remaining items or all items are bigger than A_i, return.*
4. *Search all matching items of A_i from the sequence provided by F and assign them to the stock, one by one, reducing L (or W).*
5. *Pick a non-matching item B_j which is smaller than A_i from the sequence provided by F, assign it to the bottom-left of the remaining area of A_i and locate a guillotine cutting line by the way decided by x_j described in 5.2.2). And then separate A_i into two sub-stocks with this guillotine cutting line.*
6. *Treat the first sub-stock as a temporary A_i and call steps 3 to 5 recursively; and then fill the second sub-stock by the same method until return.*
7. *If $i < m$ then $i \leftarrow i + 1$ and go to step 2, otherwise return.*

To achieve guillotine cutting, the decoder function first searches all of matching items from the sequence provided by the coding data and assigns matching items to the stock one by one. Pick a non-matching item from the sequence given by the coding data and assign it to the bottom-left of the stock. Then, locate a guillotine cutting line beside the item assigned, so that the stock will be separated into two sub-stocks. This reduces the problem to the layout of the two pieces of sub-stocks. That is, the two sub-stocks are cut into the remaining items by calling the decoder function recursively two times.

In the Introduction, the present chapter refers to five requirements on the general problem. These requirements were met in the process of decoding from an individual coding to a layout pattern.

(1) *Guillotine cutting*. Decoder function arranges the items in turn with respect to a stock until the remaining area of this stock is not suitable for any item. Then, the next piece of stock is used according to the sequence of the individual coding. A guillotine cutting line is located as soon as an item is assigned to the stock; and then the stock is cut into two pieces according to this guillotine cutting line. Thus, the problem becomes how to use two sub-stocks to create items with guillotine cutting. In this way, many guillotine cutting lines are to sub-stocks. This guarantees not only that each stock and sub-stock have at least one guillotine cutting line such that the stock can be separated, but also leads to the guillotine cutting lines as less as possible. The guillotine cutting was satisfied as well as the constraint of guillotine cutting was weakened.

(2) *Texture matching*. As the decoder function assigns and arranges items to a stock, first it must check the texture of the item against the stock according to x_i of the individual. If the textures do not match, the item is rotated 90 degrees.

(3) *Width of kerfs*. According to the *step (2)* of Section 2.3.4, each piece of stock will also be expanded by the width of kerfs before items are assigned to it. If the width of kerfs is added only to the sizes of items and not to stocks, after items are separated from the stock, some items placed along the edges of the stock will actually be bigger by half the kerfs than the real demanded size for length or width. In other studies, these items were not the exact

demanded sizes, and therefore we added *step (2)* to avoid this problem.

(4) *Adapting to strip-type stocks*. A stock is treated as a strip-type stock if its length is sufficiently long. In this case, the guillotine cutting line cannot be lengthwise. To achieve this in the algorithm all that is required is that all x_i of individuals be restricted to 0 or 1, both in the initialization of individuals and also in optimization operations at the stage of iterations.

(5) *Choosing stocks*. Individual coding provides a sequence $A_1 \ldots A_m$ for the decoder function to extract. If there are sufficient stocks, only an initial subsequence of $A_1 \ldots A_m$ will be used and not the whole sequence, and thus selection of stocks will be achieved.

6.3 Simulation Experiments

6.3.1 Comparisons of Utilization with GA

Although there have been some other studies of the stock layout problem, these do not provide much test data suitable for the present chapter. Refs. [40]-[42] [45] are about a single type of stock, whereas we need to test the multi-type case. Refs. [43] [44] are about multiple types of stocks, but they use different objective functions than the present chapter. Refs. [46]-[48] use a fundamentally different definition of the stock layout problem, making it inappropriate to compare FSO with their method.

Previously we had proposed an algorithm [51] for a

Table 6.2 General problems and results

Problems			GA						FSO					
ID	m	n	T	G	BU	AU	WU	SD	T	G	BU	AU	WU	SD
1	5	50	14.5	459	96.94	95.94	94.96	0.51	14.3	408	97.26	96.11	95.44	0.48
2	5	50	14.3	451	95.69	94.81	94.10	0.48	14.2	414	96.37	95.55	94.34	0.58
3	5	50	14.9	472	96.55	95.80	94.61	0.57	14.7	444	96.68	95.97	94.93	0.56
4	5	50	14.5	448	96.11	94.69	93.21	0.89	14.4	415	96.22	95.63	94.69	0.48
5	5	50	16.9	480	95.72	94.36	92.20	1.07	16.8	436	96.25	95.38	94.22	0.56
6	20	200	114	399	97.23	96.14	93.71	1.00	110	465	97.64	96.81	94.41	0.85
7	20	200	111	401	96.80	96.02	94.66	0.66	107	459	97.94	96.31	94.80	0.96
8	20	200	111	399	96.92	96.31	94.80	0.62	108	461	97.57	97.00	95.53	0.55
9	20	200	121	430	97.19	96.65	96.09	0.35	118	479	97.40	97.09	96.52	0.27
10	20	200	117	404	97.07	96.23	95.10	0.49	112	474	97.48	96.94	96.27	0.36
A			70.5	440	96.24	95.30	94.12	0.62	68.8	449	96.71	95.90	94.89	0.54

Table 6.3 Problems and results considering texture

Problems			GA						FSO					
ID	m	n	T	G	BU	AU	WU	SD	T	G	BU	AU	WU	SD
1	5	50	19.2	478	92.95	90.36	87.70	1.73	18.9	432	95.09	93.11	91.29	1.22
2	5	50	19.3	481	92.77	90.51	85.29	2.35	19.0	408	94.37	91.88	89.99	1.38
3	5	50	19.5	460	90.45	88.50	86.71	1.32	19.3	406	92.42	90.72	88.61	1.09
4	5	50	20.3	465	91.14	88.94	84.52	1.73	20.2	428	92.47	90.86	88.79	1.10
5	5	50	20.5	484	90.84	88.44	85.05	1.90	20.2	428	92.54	90.71	88.85	1.06
A			20.1	474	92.11	89.92	86.93	1.64	20.0	426	93.69	91.64	89.79	1.15

Table 6.4 Problems and results for all requirements

Problems			GA						FSO					
ID	m	n	T	G	BU	AU	WU	SD	T	G	BU	AU	WU	SD
1	3	50	22.1	474	94.01	93.54	92.83	0.36	21.8	406	95.49	94.88	94.48	0.35
2	3	100	72.5	486	94.92	93.24	91.81	1.15	72.4	406	96.85	95.31	94.08	0.85
3	4	50	19.1	438	92.64	91.08	89.49	0.85	19.0	361	94.94	93.66	91.99	0.81
4	4	100	76.8	490	94.81	94.26	93.73	0.37	76.6	401	95.57	95.26	94.74	0.24
5	5	50	18.3	445	93.41	92.04	89.05	1.17	18.1	366	95.83	94.41	93.32	0.67
A			49.1	471	92.99	91.82	90.53	0.76	49.0	386	94.30	93.35	92.51	0.54

large-scale rectangle stock layout problem and solved it using GA. Therefore, in order to examine the performance of FSO, we compare FSO with GA here. Since the optimization operations FSO uses are completely different from those in GA, the operations have different speeds as measured by CPU-time; therefore, we compare the iterative process of FSO and the evolutionary process of GA with respect to CPU-time rather than iterations or generations.

Hundreds of test problems were generated randomly. Each was solved using FSO and using GA 10 times using different random number seeds on a computer with a 2.00-GHz CPU clock and 2.0 GB of RAM.

Table 6.2 shows the first 10 problems and their test results under requirements (1) and (5); which are only part of 100 problems. In Table 6.2, the problems consist of two groups of five problems on the same scale. In each group, some problems have data ranges L_i,$W_i \in \{500, ..., 1000\}$, $l_i \in \{50, ..., 200\}$, and $w_i \in \{50, ..., 100\}$. The other problems have the same data range for stocks but a slightly larger data range for items: $l_i \in \{75, ..., 300\}$, and $w_i \in \{75, ..., 150\}$. In this way, the relative sizes of stocks to items vary.

Table 6.3 shows the results for the first 5 problems of Table 6.2 under requirements (1), (2), and (5). Thus, table 6.3 shows results when the same first 5 problems as shown in Table 6.2 are considered with the addition of requirement (2).

Table 6.4 considers the some problems, but with all requirements (1) to (5). In Table 6.4, the L_i is infinite which indicate strip-type stocks. $W_i \in \{500, ..., 1000\}$, l_i, $w_i \in \{100, ..., 500\}$, the textures of stocks and items are generated randomly; the width of kerfs is set as 5.

Notation used in Table 6.2 - 6.4 is as follows:

T: CPU time (s)

G: generations or iterations completed within a given CPU time

BU: best case of utilization (%) of 10 algorithm runs

AU: mean utilization (%) over 10 algorithm runs

WU: worst utilization (%) of 10 algorithm runs

SD: standard deviation

A: average for column variable over 100 or 50 problems

The algorithms always stops running at the generation or iteration at which the algorithm is completely finished, and, as shown in Table 6.2 - 6.4, FSO always finishes in less CPU-time than GA.

For BU, AU, and WU, FSO obtained the higher utilization than GA for all problems in Table 6.2 - 6.4. However the exact solution for these problems is unknown, the ability of the method may be estimated more clearly by the upper-limit of waste-rate (ULWR) rather than utilization, defined as ULWR = 100 × (1 – *Utilization*).

To the 100 problems which are reported first 10 problems in Table 6.2, GA obtains the ULWR of AU

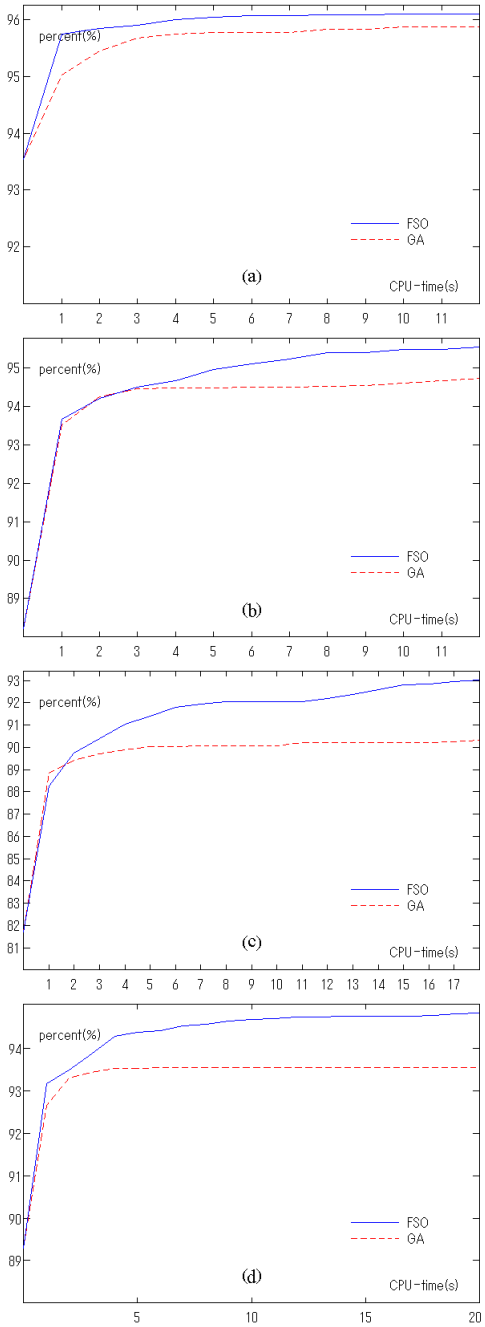

Fig. 6.2 Evolutional curves ["CPU time (s)" (×4), "Percent (%)" (×4).]

4.70 in average, and FSO obtains 4.10 in average, which shows that FSO could decrease ULWR by 12.8% in average comparing with GA. To the 50 problems which are listed first 5 problems in Table 6.3, FSO decreased ULWR by 17.1% in average, and to the 50 problems which are listed first 5 problems in

Table 6.4, FSO decreased by 18.7% in average.

Since every problem is tested 10 times with FSO and GA, the SD of percent of utilization also can be used to measure the stability of algorithm. For almost all the cases in the Table 6.2 - 6.4, it can be seen that FSO typically has a smaller SD value than that of GA.

Some test problems were chosen and the evolutionary curves with respect to CPU time were plotted, as shown in Fig.6.2. These curves show the mean percent of utilization for 10 runs. The Fig.6.2 (a) and (b) show respectively the evolutionary processes of test problems 1 and 2 of Table 6.2. The (c) and (d) figures respectively those of problem 1 of Table 6.3 and problem 1 of Table 6.4.

For our experiments, we calculated the evolutionary curves for all test problems. These figures show that FSO achieved a faster and better evolutionary process than GA, and the same for almost all the tries of the problems in the experiments. Comparing the evolutional curves by FSO with those by GA in these figures, FSO reached to the same level solution by GA in the final step within exceedingly short time, that are about 1/3 of CPU time in the Fig.6.2(b), and almost 1/10 of CPU time in the Fig.6.2(d).

6.3.2 Example of Layout Pattern

Since a picture is worth a thousand words, it is easiest to judge the quality of a layout pattern by seeing it. Although few test problems suitable for our study are available in the literature, there are some small-scale problems; we show the layout patterns here.

Ref. [52] provides a set of test data with stocks having sizes 4000 × 2900 (mm); the sizes and numbers of items as shown in Table 6.5. Fig.6.3 shows the layout pattern obtained by the algorithm proposed by Ref. [52]. Fig.6.4 shows the layout pattern obtained by FSO.

Another test problem is provided by Ref. [53]. The stocks have size 600 × 400 (mm), and the item details are given by Table 6.6. Fig.6.5 shows the layout pattern obtained by the algorithm of Ref. [53]. Fig.6.6 shows the layout pattern obtained by our proposed algorithm.

Both of these two test problems involved a single size of stock, but the number of stocks was sufficient. Also, both of these two problems required two pieces of stocks, as shown in Fig.6.3 to 6.6. In the definition of formula (1), the area R is considered as part of the

Table 6.5 Sizes and number of items from Ref. [52]

Serial Number	Length (mm)	Width (mm)	Items
1	450	60	103
2	500	100	70
3	540	70	90
4	400	100	120
5	200	100	150
6	360	150	40
7	450	120	45

Table 6.6 Sizes and number of items from Ref. [53]

Serial Number	Length (mm)	Width (mm)	Items
1	130	70	3
2	140	80	3
3	140	100	1
4	220	210	1
5	240	90	3
6	300	70	2
7	120	80	3
8	350	90	3
9	310	130	1
10	130	110	3

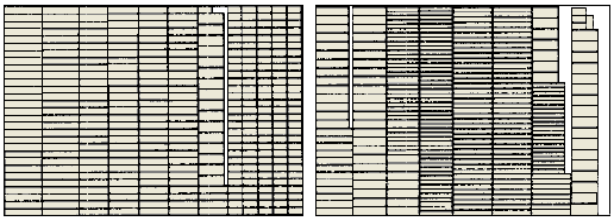

Fig. 6.3 Layout pattern from Ref. [52]

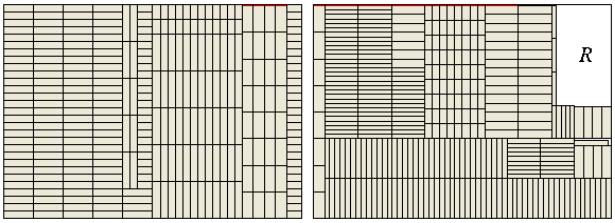

Fig. 6.4 Layout pattern by FSO for instance of Ref. [52]

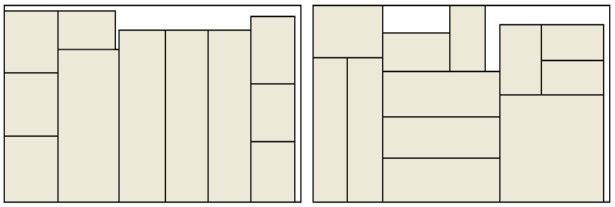

Fig. 6.5 Layout pattern from Ref. [53]

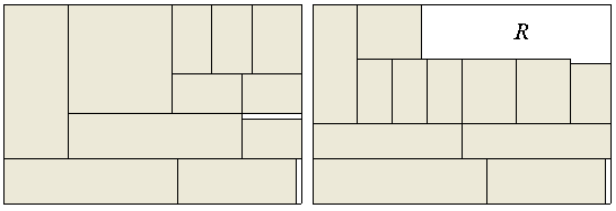

Fig. 6.6 Layout pattern by FSO for instance of Ref. [53]

objective function. This led to area *R* shown in Fig.6.4 and Fig.6.6 being as large as possible, where this area is the amount of materials remaining for use in the next project. If *R* is not considered, the areas of layout patterns shown in Fig.6.3 and Fig.6.4, as well as those in Fig.6.5 and Fig.6.6, would be considered to have the same percent of utilization of stocks. This consideration of *R* is why it is not useful to compare our test results on percent of utilization with those in prior studies.

Table 6.7 Sizes and number of items from Ref. [54]

Serial Number	Length (mm)	Width (mm)	Items
5	40	20	40
8	15	6	78
12	50	30	19
1	50	25	59
6	50	40	64
9	18	6	35
15	45	15	26
2	35	15	49
3	60	40	66
4	80	40	80

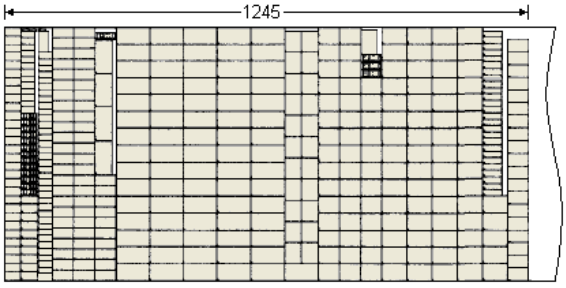

Fig. 6.7 Layout pattern from Ref. [54]

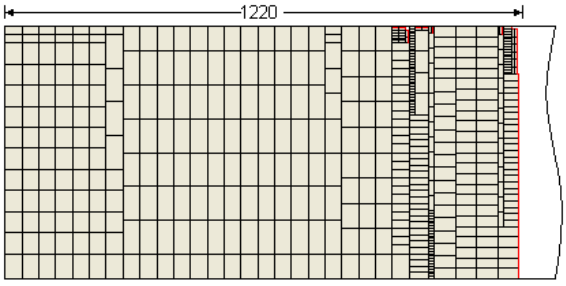

Fig.6.8 Layout pattern by FSO for instance of Ref. [54]

The above two example are only small-scale problems. For large-scale problems, it is possible that the optimized area is large enough to save whole sheets. That is, a small percentage improvement can translate into several pieces of stock saved in large-scale problems.

The third test problem is provided by Ref. [54]. The stock is strip-type stock with width 600 (mm), the item details as shown in Table 6.7. Fig.6.7 shows the layout pattern obtained by the algorithm proposed by Ref. [54], which uses the stock with length 1245. Fig.6.8 shows the layout pattern obtained by FSO which uses the stock with length 1220. It is clear that FSO obtained the solution saving material.

6.4 Discussion

In section 5.4 of last chapter we have discussed the constraints of guillotine cutting, algorithm applicability, global feature of solution and cutting efficiency of layout pattern. As the decoder function FSO adopts work under same mechanism, FSO could also minimize the constraints of the guillotine cutting. Also FSO solves the stock layout with excellent algorithm applicability, global feature of solution and high cutting efficiency of layout pattern. this section mainly discuses the FSO itself and comparing with the that FSO X.L.li proposed.

6.4.1 Computational Complexity

On the other hand, according to the original theory of FSO, choosing among the three types of operations is done by setting a priority for them. That is, first judging is carried out whether the center of individuals is better and is not too crowded; if so, then the cluster operation is performed and otherwise the algorithm continues to consider the following operation. If there is not also the better or crowded for the following operation, then the foraging operation is performed. Based on this rule, we have experimented with many test problems but could not obtain better results than GA [50]. If the condition for the clustering operation is judged first, the speed of performing the operation is slower than the operation of GA. Even optimizing all parameters of FSO for many cases, we could not obtain a significant improvement. This led us to almost abandon this optimization method.

By redesigning the process of FSO as described in Section 2.3, the effectiveness of the algorithm was much improved. In early iterations, most individuals performed the foraging operation; this avoids calculating the center point, which has a high cost in terms of CPU-time. In the later iterations, more individuals performed the clustering or following operations; this FSO is effective at finding better solutions despite the CPU-time costs. In the experiments involving small-scale problems, such as described in Section 3.2, the speed of iterations of

FSO is slower than the evolution of GA. Despite this, the FSO still obtained a higher optimization curve than GA. For larger-scale problems, as in Section 3.1, the speed of FSO catches up to or even exceeds that of GA.

6.4.2 Definition of Center Position

There are many optimization algorithms; the reason why FSO is employed in this application are its global feature, few parameters and easy to implement.

X.L. Li gave an example of the traveling salesman problem (TSP) in his doctor thesis. The coding method of TSP is also a sequence of nodes, similar to that of the stock layout problem. But he defines the distance between two individuals as the number of different points (columns) between two individuals. This definition does not reflect the approximate difference between two solutions in the stock layout problem, so we redefine it as shown in (4).

This chapter proposed an approach based on FSO to solve the two-dimensional guillotine cutting problems. The proposed algorithm improved the FSO with new operational rule, which perform the iteration so quickly and purposefully that the performance of FSO was improved distinctly. Also, an encoding method and decoder function were proposed in this chapter, which achieve the layout pattern with local optimum based on the individual encoding. FSO with this decoder function actually works combining local optimization and global optimization. Large number of tests indicated the superiority of proposed algorithm.

Chapter 7

Discussion

Based on the contents stated in previous chapters, this charter firstly extends the problem of lethal chromosomes for genetic algorithms to solve the combinatorial optimization problems into the infeasible solution of evolutionary computation solving the combinatorial optimization problems. Secondly, the chapter analyzes the fact that the infeasible solution is inevitable and the importance of treating the infeasible solution for evolutionary computation. On the other hand, this chapter also compares the fish swarm optimization with genetic algorithm and other evolutionary computation, analyzes their characteristics separately. Finally the chapter discusses application of combinatorial optimizations problems.

7.1 Evolutionary Computation and Infeasible Solutions

As the season why the lethal chromosomes appear in GA, evolutionary computation also face the infeasible solutions in solving the combinatorial optimization problems. The method of GA handling the lethal chromosomes can be referenced by evolutionary computation to handle the infeasible solutions.

7.1.1 The Universality of Infeasible Solutions

In evolutionary computation, either evolutionary algorithms such as genetic algorithm (GA) and immune algorithm (IA) or swarm intelligence algorithms such as fish swarm optimization (FSO) and particles swarm algorithm (PSO), they all need to encode the characteristics of solution into chromosome as optimization object for solving the combinatorial optimization problems. This is main unique characteristic of evolutionary computation comparing with traditional algorithm.

A popular definition of combinatorial optimization problems is that: within discrete, limited mathematical structures, one (or asset of) feasible solution to be obtained with maximizing (or minimizing) the object function, simultaneously the solution obtained has to satisfy the constraints condition of problem. In the other word, constraints condition is necessary for constraints condition.

Since it is inevitable that evolutionary computation encodes the characteristics of solution into chromosome as optimization object and also constraints condition of combinatorial optimization problems, evolutionary computation will often encounter the infeasible solution in solving the combinatorial optimization problems. Of course it is not always to appear the infeasible solution; whether there is a feasible solution depends on the encoding and decoding method. Next, we explain the reason why appearing the infeasible solutions and the circumstances without infeasible solutions.

As previous description, those two algorithms, proposed in chapter 3 and chapter 4, involve the infeasible solutions and also lethal chromosomes. In chapter 3, the chromosome is an *n*-bit 0-1 string. If we changed the encoding method that chromosome become a permutation of objects, decoder function also changed that the objects are selected in turn according to the permutation chromosome provides until knapsack was full without overload, there was no any lethal chromosomes appearing. In chapter 4, the chromosome is an *n*-bit multi-value string. If the chromosome was changed as two rows that first row is permutation of knapsacks, the second row is permutation of objects, the decoder process was changed that the knapsacks were filled one by one in turn with objects also in turn according to permutations chromosome provides until every were fill without over load, there was also no any lethal chromosomes.

However this encoding and decoder method mentioned above will lead the encoding space expand extensively, and maybe different chromosomes correspond to same solution. That will cause the algorithm with calculation of redundancy.

The individual encoding systems described in chapter 5 and chapter 6 can also be changed to

generate the infeasible solution. For example, stocks were regarded as knapsacks and the items were regarded as objects, stock layout problem would be translated to the multi-knapsack problem, which was solved with encoding system described in chapter 4, the infeasible solution would appeared.

In evolutionary computation, the emergence of infeasible solution decides by the encoding system. Encoding system could be changed to avoid from infeasible solutions. However encoding system should be dominated by factors that benefit performance of algorithm instead of avoiding from infeasible solution. Almost all combinatorial optimization problems have the constraints conditions; problem of infeasible solution is inevitable in solving optimization problems with evolutionary computation. Flexible strategies have significance.

7.1.2 The Strategies to Handle the Infeasible Solutions

In any GA implementation for constrained optimization problems, it is an important issue to handle constraints which may generate the infeasible solutions. A number of procedures were described which considered the constraint in an optimization problem. Zbigniew Michalewicz [14] (1996) presented a suitable classification of these procedures which are described (i) rejecting strategy, (ii) repairing strategy, (iii) modification of genetic operators, (iv) penalizing strategy. Because lethal chromosomes with evolution contain the excellent schemas in spite of the chromosomes being lethal, this thesis proposes the repairing strategy to handle and recycling using the lethal chromosomes.

In the chapter 3 and chapter 4, we proposed two different handling operation to lethal chromosomes despite that we called them both immune operations. With proposing the two type of method to handle the lethal chromosome, actually we would like to demonstrate two types of ideas to handle infeasible solutions with repairing strategy. The two methods could be exchanged to use each other between chapter 3 and chapter 4.

We assume that the immune operation described in chapter 3 was applied to the problem MKP of chapter 4. The vaccine would no longer an n-bit string, but an $(m + 1) \times n$ matrix referred as $S_{[(m+1)\times n]}$. The value of $S_{[ij]}$ indicates the times of j^{th} object being selected into i^{th} knapsack ($i = 1, 2 \ldots m$; $j = 0, 1 \ldots n - 1$), $S_{[ij]}$ at case $i = 0$ indicates the times of j^{th} object without being selected into i^{th} knapsack. Vaccination could be achieved that removing the some objects from overloading knapsacks according to small value in $S_{[(m+1)\times n]}$.

On the contrary, the idea of immune operation described in chapter 4 also can be applied to the problem of chapter 3. The method being applied on multi-value encoding was translated to be applied on bin- value encoding; the detail would be more simplified.

The two ideas we proposed not only could be introduced different problems but also could be introduced form GA to other evolutionary computations. The problem of infeasible solutions for evolutionary computations could be still improved so much; this thesis just completes a little.

7.2 Complexity of the Algorithm

Computation complexity includes the time complexity and space complexity. There are some factors impacting the computations complexity on evolutionary computations, iteration of population, optimal operation and calculating the fitness etc, especially calculating the fitness. The time complexity of the evolutionary computations solving the combinatorial optimization problems is $O(G \bullet P \bullet n)$, where G is the generations, P if population and n is the length of encoding. The space complexity is $O(P \bullet n)$.

This thesis proposed an IGA with immune operation in the chapter 3-6 which burdens to the time complexity and space complexity on GA. This was also the cost to improve the effectiveness of algorithm. For determining this cost comparing with improvements of effectiveness, previous chapters measured the results against to CPU-time. Section 3.3.2 lists the CPU-time within that the best solution were obtained. Although 4.3.1 showed the evolutionary curve based on generation instead of CPU-time, at last it still also given the results about CPU-time. Section 6.3.1 also listed large number of result data based on CPU-time.

We could learn from such much experiment results based on CPU-time that proposed algorithm could obtained the better solution with shorter CPU-time. In the other word, improvements of defectiveness have covered and exceeded the burdens of computations complexity. These indicate that method of improving the algorithm at cost of burdening the computations complexity is feasible to evolutionary computations.

7.3 Application of Combinatorial Optimizations Problems

Combinatorial Optimization is an old as well as young subject which is applied in many fields. In the late 29th century, along with industrial technology revolution and the development of modern management science, especially the rapid progress of computer technology and extensive application in various industries, combinatorial optimization has been expanding into a branch of computer science and operations research. Some issues and approaches hundreds of years ago mathematicians occasionally think of already played an important role in network communications, logistics management, transport planning, etc. All of these indicated the great prospects of this subject.

The description of the combinatorial optimization problem is very simple, and they have the strong engineering representative, but it is difficult to solve the most optimal solution as their feature of "combinatorial explosion". To one of the combinatorial optimization problems, NP-hard problems, researchers often focus on finding out a feasible solution in a relatively short time (polynomial time).

In fact, heuristic algorithm has a large number of literatures and a wide range of applications. But heuristic algorithm has a fatal weakness that the solutions have no any guarantee with quality. To any heuristic function, there are always some instances to failure, that is, the deviation from the optimal solution for far. Approximation algorithm fetches up for lack of heuristic algorithms, it can define the quality of solution. Approximation algorithm is also a class of heuristic algorithms, but it is limited by the upper bound of ratio of approximate solution and optimal solution. In this case, evolutionary computations have a widely application to solve the combinatorial optimization problem.

Chapter 8

Conclusion and Prospects

In this study, we firstly introduced the basics concept of evolutionary computation and combinatorial optimization problems and two different methods to handle the lethal chromosomes of genetic algorithms on two different problems multi-dimensional knapsack problems and multi-knapsack problems. Next, a genetic algorithm and fish swarm optimization were designed severally to solve the stock layout problem with guillotine cutting and other process requirements. Finally, we boldly conceive that proposed ideas could be introduced to other where even about evolutionary computations to solve the combinatorial optimization problems.

8.1 Conclusion of the Thesis

In GA solving the MDKP, This study discovered an important feature of lethal chromosomes, with that we designed an algorithm IGA working on the double island model to use this feature of lethal chromosomes, which introduced the immune idea into GA. Appling IGA to MDKP, a large number of testing results indicate that using lethal chromosomes based on immune operation could obviously improve search performance of GA.

To lethal chromosomes of GA solving the MKP, the thesis tried a completely different immune operation on double island model IGA. Experimental results with evolutionary curve showed that this method could also obviously improve the evolutionary performance of GA solving the MKP. If proposed method would be improved again in decreasing the time complexity, GA will be applied in more fields.

To the stock layout problems with guillotine cutting, this thesis proposed a two-line chromosome decoded by a heuristic method and optimized by GA and FSO. Both decoder function and optimal algorithms GA and FSO are the effective way to solve the problem. Especially to FSO which with a new operational rule, could perform the iteration so quickly and purposefully that the performance of FSO was improved distinctly.

The encoding and decoder system have a good local searching based on the chromosome, the GA and FSO are suitable for global searching. The combination of them can figure out layout of large scale stock layout problems effectively. The results of computational simulation showed that these designs could obtain a satisfactory solution easily.

These studies are only the tip of the iceberg for evolutionary computation to calculate the combinatorial optimization problems and even operation researches. There is still muck works should be done in the future.

8.2 Future Works

In this study, the cost of using the lethal chromosomes in GA is having burdened the complexity of algorithm despite that can be fetched up by improvements of performance. In the future, if GA is improved on either complexity of operation or algorithm model, the algorithms will be applicable to a wider range of field.

In addition, this thesis studied to handle the lethal chromosomes of GA only on two combinatorial optimization problems MDKP and MKP; there are many problems for GA to face the infeasible solution and lethal chromosomes. GA should be solved the problem of lethal chromosomes well on also other constraints optimization problems.

On the other hand, there have been some new artificial excellent algorithms in recent years such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and FSO. To these new algorithms, much algorithm mechanism should be improved. For example in FSO, some researchers are studying the problems of fish swarm following each other. Besides these, the validity and stability of convergence, deeply researches for mechanism of fish swarm foraging are to be completed.

8.3 Prospects of Evolutionary Computation

With developments over the past decades,

evolutionary computation has developed into a new intelligent processing technology, has received more and more attention from various fields. However as a young intelligent processing technology, evolutionary computation has not yet been enough for perfection and maturity. But in the 21st century, its development is unstoppable.

To various evolutionary computations, their working mechanism, mathematical basis and dynamics features, etc, will be studied and developed deeply. That will not only contribute to analyze the algorithm properties, improve the existing algorithms but also advance to design new algorithms.

With the artificial intelligence being required and applied increasingly widespread, based on studying and improving existing algorithms, and under the context of various subjects cross-developing, the new excellent simulation algorithms will become the researching focus.

The integration of various intelligent algorithms will also become a new researching focus. Every evolutionary computations has strengths and weaknesses, if various algorithms be combined into a complementary, coordinate and advanced system, the computing power and effectiveness will be unexpected.

References

[1] Bagley J D, “The behavior of adaptive systems which employ genetic and correlation algorithms”, Dissertation Abstracts International, 28(12), 1968.

[2] Holland J H, “Adaptation in Nature and Artificial System”, Ann Arbor: Univ. of Michigan Press, 1975.

[3] Schwefel H P, “Numerical Optimization of Computer Models”, Chichester:John Wiley, 1981.

[4] G Beni, “The concept of cellular robotic systems”, In Proceedings of the IEEE International Symposium on Intelligent Systems, pp.57-62, IEEE Press, Piscataway, NJ, 1988.

[5] Iima Hitoshi, Sannomiya Nobuo, “The Influence of Lethal Gene on the Behavior of Genetic Algorithm”, The society of instrument and control engineers, vol.31, no.5, pp.569-576, 1995.

[6] Mengchun Xie, Tetsushi Yamaguchi, Tomohiro Odaka, Hisakazu Ogura, “An Analysis of Evolutionary States in the GA with lethal Genes”, The Information and Systems Society, Institute of Electronics, Information and Communication Engineers, vol.J79-D-II, no.5, pp.870-878, 1996.

[7] P.C. Chu and J.E. Beasley, “A Genetic Algorithm for the Multidimensional Knapsack Problem”, Journal of heuristics, vol.4, no.1, pp.63-86, June 1998.

[8] Günther R. Raidl, “Weight-Codings in a Genetic Algorithm for the Multiconstraint Knapsack Problem”, Proceedings of the 1999 ACM symposium on applied computing, Texas, United State, no.02, pp.291-296, February 1999.

[9] Farhad Djannaty, Saber Doostdar, “A Hybrid Genetic Algorithm for the Multidimensional Knapsack Problem”, Int. J. Contemp. Math. Sciences, vol. 3, no.9, pp.443-456, 2008.

[10] P. C. Chu, J. E. Beasley, “A genetic algorithm for the generalised assignment problem”, Computers and operations research, vol.24, no.1, pp.17-23, January 1997.

[11] Günther R. Raidl, “An Improved Genetic Algorithm for the Multiconstrained 0–1 Knapsack Problem”, Proceedings of the 5th IEEE International Conference on Evolutionary Computation, Anchorage, AK, pp.207-211, May 1998.

[12] Alex S. Fukunaga, “A New Grouping Genetic Algorithm for the Multiple Knapsack Problem”, 2008 IEEE Congress on Evolutionary Computation (CEC 2008), Hong Kong, China, pp.2225-2232, June 2008.

[13] Alex S. Fukunaga, Satoshi Tazoe, “Combining Multiple Representations in a Genetic Algorithm for the Multiple Knapsack Problem”, 2009 IEEE Congress on Evolutionary Computation (CEC 2009), Trondheim, Norway, pp.2423-2430, May 2009.

[14] Michalewicz. Z, “Genetic Algorithm + Data Structurs = Evolution Programming”, Springer Verlag, 3rd edition, 1996.

[15] Jens Gottlieb, “Permutation-Based Evolutionary Algorithms for Multidimensional Knapsack Problems”, Symposium on Applied Computing Proceedings of the 2000 ACM symposium on Applied computing, Como, Italy, vol.1, pp.408-414, March 2000.

[16] V. Gabrel, M. Minoux, “A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems”, Operations research letters, vol.30, no.4, pp. 252-264, August 2002.

[17] Maoguo Gong, Licheng Jiao, Wenping Ma, Shuiping Gou, “Solving multidimensional knapsack problems by an immune-inspired algorithm”, Evolutionary Computation, Issues.25-28, pp.3385-3391, Sept 2007.

[18] Guan-Chun Luh, Chung-Huei Chueh, “Multi-objective optimal design of truss structure with immune algorithm”, Computers and structures, vol.82, Issues.11-12, pp.829-844, May 2004.

[19] Lei Wang, Licheng Jiao, “The immune genetic algorithm and its convergence”, Fourth International on Signal Processing Proceedings (ICSP 1998), Beijing, China, vol.2, pp.1347-1350, October 1998.

[20]Ma Liang, Wang Longde, “Ant Optimization Algorithm for Knapsack Problem[J]”, Journal of Computer Applications, vol.21, no.8, pp.4-5, 2001.

[21]JaoLiCheng, PanJin,WangLei, “The Immune Algorithm[J]”, Acta Electronica Sinica, vol.28, no.7, pp.74-78, 2000.

[22]JiaoLiCheng, DuHaiFeng, “Development and Prospect of the Artificial Immune System[J]”, Acta Electronica Sinica, vol.31, no.10, pp.1540-1548, 2003.

[23]XuXiaoDong, LiCongXin, “Application of immune genetic algorithm in job-shop scheduling problem[J]”, Journal of Southeast University (Natural Science Edition), vol.36, no.3, pp.437-441, 2006.

[24]FanJianHua, WangXiuFeng, “Vehicle Routing Optimization Problem Based on Immune Algorithm[J]”, Computer Engineering and Applications, vol.42, no.4, pp.210-212, 2006.

[25]SunYongFei, “An Artificial Immunity Algorithm of the QoS Multicast Routing Problem[J]”, Computer Engineering and Applications, vol.42, no.11, pp.131-134, 2006.

[26]MARÍA A. OSORIO, FRED GLOVER, PETER HAMMER, “Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions”, Annals of Operations Research 117, 71-93, 2002,©2003 Kluwer Academic Publishers, Manufactured in The Netherlands.

[27]Ma Xuan, Zhang Yalong, Li Shengmin, “An equal apothem development algorithm for hyperboloid metal wall[J]”,. Computer Engineering and Applications, 45(23): 217-235, 2009.

[28]http://search.sipo.gov.cn/sipo/zljs/hyjs-yx-new.jsp ?recid=CN200710017662.X&leixin

[29]Cao Ju, Feng Song, “The application of genetic algorithm in rectangular object optimal layout [J]”, Computer Engineering and Application, vol.5, no.2, pp.5-7, 1999.

[30]Yang Wei, Luo Yang, LIU Shengqing, “Genetic algorithm for large scale rectangular object optimal embedplacement[J]”, Journal of Sichuan University (Engineering science edition), vol.33, no.5, pp.59-62, 2001.

[31]Han Xijun, Ding Genhong, “The optimum packing of rectangles based on improved genetic algorithm[J]”, Computer Engineering and Application, vol.42, no.25, pp.63-65, 2006.

[32]Luo Yiping, Liu Jun, Li Bing, Jiang Zhuangde, “A practical heuristic algorithm for rectangle part s packing problem[J]”, Journal of Engineering Graphics, vol.24, no.4, pp.50-58, 2003.

[33]Wang Huachang, Tao Xianwei, Li Zhigang, “A comprehensive algorithm for the layout of large scale rectangular parts”, Journal of Huazhong University of Science and Technology, Nature Science, vol.31 no.6.Jun, 2003.

[34]Xing Wenxun, Xie Jingxing, “Modern Optimization Computational Method[M]”, (second edition), BeiJing: Qinghua University Press, 2005.

[35]Hopper E. and Turton B. C. H., 2002, “An empirical study of meta-heuristics applied to 2D rectangular bin packing”, vol.2, no.1. ISBN 2-912590-13-2, ISSN Regular 1625-7545.

[36]http://people.brunel.ac.uk/~mastjjb/jeb/info.html

[37]Stefan Jakobs, “On genetic algorithms for the packing of polygons”, European Journal of Operational Research 88(1996), pp.165-181, 1996.

[38]E.Hopper, B.Turton, “A Genetic Algorithm for 2D Industrial Packing Problem”, Computers & Industrial Engineering 37 (1999), pp.375-387, 1999.

[39]Marc van Kreveld, “Bettina Speckmann. On rectangular cartograms”, Computational Geometry 37 (2007), pp.175-187, 2007

[40]N. Christofides, E. Hadjiconstantinou, “An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts”, European Journal of Operational Research, vol.83, no.1, pp.21-38, 1995.

[41]V. Parada, M. Sepulveda, M. Solar, “A Gomes: Solution for the constrained guillotine cutting problem by simulated annealing”, Computers & Operations Research, vol.25, no.1, pp.37-47, 1998.

[42]V. Parada, R. Palma, D. Sales, A. Gomes, “A comparative numerical analysis for the guillotine two-dimensional cutting problem”, Annals of Operations Research, vol.96, no.1-4, pp. 245-254, 2000.

[43]R. Morabito, M. N. Arenales, “Staged and constrained two-dimensional guillotine cutting problems: An AND/OR-graph approach”, European Journal of Operational Research, vol.94, no.3, pp.548-560, 1996.

[44] V. Mornar, B. Khoshnevis, “A cutting stock procedure for printed circuit board production”, Computers & Industrial Engineering, vol.32, no.1, pp.57-66, 1997.

[45] A. Bortfeldt, “A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces”, European Journal of Operational Research, vol.172, no.3, pp.814-8371, 2006.

[46] G.-C. Lee, Y.-D. Kim, “Algorithms for adjusting shapes of departments in block layouts on the grid-based plane”, The International Journal of Management Science, vol.28, no.1, pp.111-122, 2000.

[47] Y. D. Cui, “Dynamic programming algorithms for the optimal cutting of equal rectangles”, Applied Mathematical Modelling, vol.29, no.11, pp.1040-1053, 2005.

[48] S. Tiwari, N.Chakraborti, “Multi-objective optimization of a two-dimensional cutting problem using genetic algorithms”, Journal of Materials Processing Technology, vol.173, no.3, pp.384-393, 2006.

[49] E. Ackerman, G. Bareque, R. Y. Pinter, D. Romik, “The number of guillotine partitions in d dimensions, Information Processing Letters”, vol.98, no.4, pp.162-16731, 2006.

[50] X.L. Li, Z.J. Shao, J.X. Qian, “An Optimizing Method Based on Autonomous Animats: Fish-swarm Algorithm”, Systems Engineering - Theory & Practice, vol.006, no.11, pp.32-38, 2002.

[51] Y.L. Zhang, H. Ogura, X. Ma, J. Kuroiwa, T. Odaka, “An optimization method of large scale stock layout with guillotine cutting”, The 6th International Conference on Modeling Decisions for Artificial Intelligence, Awaji Island, Japan, pp.199-207, November 2009.

[52] H.C. Wang, X.W. Tao, Z,G. Li, “A synthetical algorithm for the optimal layout of rectangular part”, Journal of Huazhong University of Science and Technology (Nature Science Edition), vol.31, no.6, pp.9–12, 2003.

[53] X.W. Tao, H.C. Wang, Z.G. Li, “Optimal solution of rectangular part layout based on rectangle filling algorithm”, China Mechanical Engineering, vol.14, no.13, pp.1104–1108, 2003.

[54] C. Yang, J.Y. Shi, H.M. Gu, “Packing of rectangular using genetic simulated annealing algorithm”, Journal of Qingdao University of Science and Technology, vol.25, no.5, pp.452-456, 2004.

[55] X.Y. Tu, “Artificial animals for computer animation: biomechanics, locomotion, perception, and behavior”, Springer-Verlag Berlin, Heidelberg ©1999, ISBN:3-540-66939-6.

[56] Yalong Zhang, Xuan Ma, Jousuke Kuroiwa, Tomohiro Odaka, Hisakazu Ogura, “A Genetic Algorithm with Utilizing Lethal Chromosomes”, 2009 IEEE International Conference on Fuzzy Systems, Jeju Island, Korea, pp.2047-2050, August 20-24, 2009.

Acknowledgements

The past three years is my hard, harvesting and happy three years in my life, left to me many unforgettable memories. In these years, I received much assistance from many individuals and institutions, without these selfless help I am not able to complete my doctoral course. So I'd like to express my most sincere thanks here.

I should firstly thank my instructor Prof.Hisakazu Ogura. In these three years, my instructor gave me help too much either in learning or in life. When I first came to Japan, with considering that I were not familiar with the situation in Japan yet that time, the instructor arranged some students to prepare accommodation in advance and even let them go to airport to pick me up. In aspect of learning, instructor insisted to give me guidance once for every week even though he was always busy. In order to guide me to understand an academic issue, he often collected a lot of information including books, papers and research reports to me, and patiently explained to me. Instructor have commented and revised almost every paper I wrote; it was impressive to me that he could always give me some deeper insights to modify the papers. On the other hand, instructor provided me many opportunities to take part in various academic conferences for training me to make the presentation; he even personally took me go to some conferences. In these three years, my instructor taking me has been to many places, Jeju Island, Hiroshima, Hang Zhou, Ningbo, Shanghai even Shao Xing and Zhou Shan etc. Instructor also looked after me too much in life, gave me opportunity to be a tutor for new student etc. In anyway, I should not only thank my instructor but also respect him and study from him.

I should also thank the Associate Prof.Jousuke Kuroiwa who is an instructor of laboratory where I study. Prof.Kuroiwa gave me guidance at research and papers once for every week despite that he was busy with many students need being supervised by him. He played an important role in one of my papers being published. In the process of that paper being completed, Prof.Kuroiwa patiently proposed the comments time and time for me to revise the manuscript, even personally wrote the cover letter and submit the manuscript to journal for me. For these Prof.Kuroiwa gave me, I will never forget.

I'd like to thank Prof.Tomohiro Odaka who is a professor of University of Fukui. In the past three years, Together with Prof.Ogura and Prof.Kuroiwa, Prof.Odaka attended to the doctoral seminar for every Monday and gave suggestion to our several doctoral students. In order to comment the paper to me, it was most impressive that he printed my manuscript and marked the commented with red, then gave to me when he met me in the elevator. I express my thanks here to him.

Associate Prof.Xuan Ma is an important person in my life, who is now an Associate Professor of Xi'an University of Technology. He is my previous supervisor when I completed my master's course in Xi'an University of Technology. He not only has taught me the knowledge, but also has ever helped me in the past when I faced the difficult for my family member. It is also him introduced me to Prof.Ogura as a doctoral student. I deeply thank him.

The International Student Division of University of Fukui provided much supports to me during my doctoral course. Before I came to Japan for doctoral course, Ms.Hayashi connected with me by e-mail for many times for admission procedures, she fervently reminded me too much for what I should prepare in advance. In the later after I came to Japan, she and other staffs of International Student Division provided a lot of enthusiasm services to me. I'd also like to thank them very much.

I express my heartfelt thanks to other staff and students in our laboratory. I thank Mr.Dai Geng and Mr.Liang Wang for having gone to Komatsu Airport to pick me up when I first came to Japan. And also I should thank Mr.Jian Zhou, Mr.Jiehang Deng and Ms.Matsumura for having helped me so much. Mr.Hamata and other enthusiastic students of our laboratory always opportunely supported me to debug

my computer, I heartfelt thank them.

The dear Mr.Yoshio Hida selflessly supported me to study with peace of mind; he sent to me the fresh vegetables, rice and other items in daily using for countless times during these three years. In my surprise, with learning I was not used to Japanese tatami, he have carried a wood bed to me from his home when a rainy day. For all of these, I will never forget in my heart.

I will never forget my dear friends Ms.Fuji and Miss Saki who live in Fukui, I thank them so much for having brought out much good time and happy memories for my wife and me in Fukui.

I thank Ms.Akado for teaching my wife Japanese weekly for volunteering and giving care of us in life, I thank her so much.

I Thank Mr.Jianxin Ai, his family and his relatives for much care and assistance. They have ever helped me deeply in the past.

Thank my parents and family for understanding and supporting.

I especially thank for my wife, Ms.Xuemei Li. She gave up her job for coming to Japan together with me. She is now responsible for daily household chores so dedicated that I have time to concentrate on writing my thesis. Originally my wife is a college vocal teacher, she could live very well in China, but she is now hard in Japan as language barrier. For all I can give an account of her is to take care of her well and pay more cherishing to her.

Finally, I thank the Japanese Government (MONBUKAGAKUSHO: MEXT) Scholarship provided by MEXT (Ministry of Education, Culture Sports, Science and Technology of Japan), with which I have been able to concentrate my main energy on researches and complete my doctoral course.

Publication List

Journals

[1] Yalong Zhang, Hisakazu Ogura, Jousuke Kuroiwa, Tomohiro Odaka, “Fish Swarm Optimization Method for the Two-Dimensional Guillotine Cutting Problem”, Journal of Signal Processing, vol.15, no.3, pp.225-234, May 2011.

[2] Yalong ZHANG, Hisakazu OGURA, Xuan MA, Jousuke KUROIWA, Tomohiro ODAKA, “A genetic algorithm using lethal chromosomes based on immune operation for multidimensional knapsack problem”, The Institute of Electronics, Information and Communication Engineers (IEICE). (in submission)

[3] MA Xuan , ZHANG Yalong, “A genetic algorithm for the layout of large scale rectangular parts”, CAAI Transactions on Intelligent Systems, vol.2, no.5, pp.48-52, 2007, ISSN:1673-4785, CNSN: 23-1538/TP. (in Chinese)

[4] MA Xuan, SUN Limin, ZHANG Yalong, “Genetic Algorithm for degree-constrained QoS multicast routing”, Computer Engineering and Applications, vol.43, no.9, pp.114-116, 2007, ISSN: 1002-8331, CNSN:11-2127/TP. (in Chinese)

[5] MA Xuan, ZHANG Yalong, ZHAO Dou, “Method of using lethal chromosome of genetic algorithm”, Computer Engineering and Applications, vol.43, no.10, pp.38-40, 2007, ISSN:1002-8331, CNSN:11-2127/TP. (in Chinese)

[6] MA Xuan，ZHANG Yalong，LI Shengmin, “Equal apothem surface development method for hyperboloid metal wall”, Computer Engineering and Applications, vol.45, no.23, pp.217-235, 2009, ISSN:1002-8331, CNSN:11-2127/TP. (in Chinese)

International Conferences

[7] Yalong Zhang, Xuan Ma, Jousuke Kuroiwa, Tomohiro Odaka, Hisakazu Ogura, “A Genetic Algorithm with Utilizing Lethal Chromosomes”, 2009 IEEE International Conference on Fuzzy Systems, Jeju Island, Korea, pp.2047-2050, August 20-24, 2009.

[8] Yalong Zhang, Hisakazu Ogura, Xuan Ma, Jousuke Kuroiwa, Tomohiro Odaka, “An Optimization Method of Large Scale Stock Layout with Guillotine Cutting”, The 6th International Conference on Modeling Decisions for Artificial Intelligence, Awaji Island, Japan, pp.199-207, November 30 - December 2, 2009.

[9] Xuan Ma, Limin Sun, Yalong Zhang, “A Two-Stage Genetic Algorithm for the Multi-multicast Routing”, Second International Symposium, ISICA 2007, Wuhan, China, pp.204-213, September 21-23 2007, LNCS 4683, Springer-Verlag Berlin Heidelberg 2007.

Domestic Conferences

[10] Yalong Zhang, Jousuke Kuroiwa, Xuan Ma, Tomohiro Odaka, Hisakazu Ogura, “A Genetic Algorithm for Multidimensional Knapsack Problem”, Joint Conference of Hokuriku Chapters of Electrical Societies 2009（平成21年度電気関係学会北陸支部連合大会）, Japan Advanced Institute of Science and Technology, September 12-13, 2009. (優秀論文賞)

[11] Yalong Zhang, Hisakazu Ogura, Xuan Ma, Jousuke Kuroiwa, Tomohiro Odaka, “An Immune Operation for Lethal Chromosomes of Genetic Algorithm”, Information Processing Society of

Japan 50th anniversary and 72nd National convention of IPSJ（情報処理学会創立50周年記念（第72回）全国大会）, University of Tokyo (Hongo Campus), pp.405-406, March 9-13, 2010.

[12] Yalong Zhang, Hisakazu Ogura, Jousuke Kuroiwa, Tomohiro Odaka, “Soft Optimization Method with Swarm Intelligence for Large Scale Stock layout Problems”, Fuzzy System Symposium 2010（第26回ファジィシステムシンポジウム）, University of Hiroshima, pp.21-26, September 13-15, 2010.

[13] Jing Guo, Yalong Zhang, Hisakazu Ogura, Jousuke Kuroiwa, Tomohiro Odaka, Izumi Suwa, Haruhiko Shirai, “Recycling Lethal Chromosomes Based on Immune Operation in Genetic Algorithm for Multi-Knapsack Problem”, Fuzzy System Symposium 2011（第27回ファジィシステムシンポジウム）, University of Fukui, September 12-14, 2011.